

E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Proyectos electrónicos con microcontrolador PIC16F877A



Grado en Ingeniería
en Tecnologías Industriales

Trabajo Fin de Grado

Autor: José Javier Imas González

Director: Carlos Ruiz Zamarreño

Pamplona, a 27 de Junio de 2017

Abstract

This work comprises the development and implementation of 3 electronic projects based on microcontroller PIC16F877A. Each project has consisted of the following stages: proofs phase and programming, schematic design, design and manufacture of the printed circuit board (PCB), assembly of the components on the PCB, bringing into operation and design of a casing using 3D software.

The first project, in which the programming plays a major role, is an alarm-clock based on I²C communication between the microcontroller and the RTC (Real Time Clock) DS3231 that also uses a 2x16 LCD display.

The remaining projects have a more practical approach. The second project is a toy consisting of two servomotors and a laser diode individually controlled by a joystick. The last project is a 3x3x3 cube of LEDs RGB (Red, Green, Blue). Each LED's illumination (81 in total) is controlled independently.

Resumen

El presente trabajo fin de grado comprende el desarrollo e implementación de 3 proyectos electrónicos basados en el microcontrolador PIC16F877A. Cada proyecto ha constado de las siguientes etapas: fase de pruebas y programación, diseño del circuito electrónico, diseño y fabricación del circuito impreso (PCB), montaje de los componentes sobre el circuito impreso, puesta en funcionamiento y diseño de una carcasa mediante software 3D.

El primer proyecto, en el que la programación cobra mayor importancia, es un reloj-despertador basado en la comunicación I²C entre el microcontrolador y el RTC (Real Time Clock) DS3231 y que utiliza una pantalla LCD 2x16.

Los dos proyectos restantes poseen un carácter más práctico. El segundo proyecto es un juguete consistente en dos servomotores y un diodo láser controlados de forma autónoma mediante un joystick. El último proyecto es un cubo de LEDs RGB (Red, Green, Blue) 3x3x3 en el que la iluminación de cada uno de los 81 LEDs se controla de forma independiente.

Keywords

- PIC Microcontroller
- PCB
- RTC
- Servomotor
- LED

Palabras clave

- Microcontrolador PIC
- PCB
- RTC
- Servomotor
- LED

Índice

0	Introducción.....	3
1	Método de trabajo y programas utilizados	4
2	Aspectos comunes a todos los proyectos	7
2.1	Diseño del esquemático y componentes utilizados	7
2.2	Diseño del PCB.....	9
3	Reloj Despertador.....	11
3.1	Introducción	11
3.2	Diseño esquemático.....	11
3.2.1	Esquemático	13
3.2.2	Listado de componentes	14
3.3	Módulo DS3231	15
3.3.1	Registros de configuración	15
3.3.2	Registros tipo BCD.....	16
3.3.3	Registros tipo hora.....	18
3.3.4	Registros de las alarmas	20
3.3.5	Registros de temperatura	21
3.3.6	Comunicación I ² C	22
3.4	Configuración del microcontrolador	24
3.4.1	Configuración de los puertos	24
3.4.2	Configuración TMR0	25
3.4.3	Configuración TMR1	25
3.4.4	Módulo CCP1	27
3.4.5	Módulo CCP2.....	28
3.5	Funcionamiento básico	29
3.5.1	Cómo configurar el reloj (TIME SET)	29
3.5.2	Cómo configurar la alarma (ALARM SET).....	29
3.5.3	Cómo apagar la alarma	31
3.6	Cálculo de del día de la semana.....	32
3.7	Programa fuente	33
3.8	PCB	40
3.9	Diseño 3D	42
3.10	Multimedia	44
4	Laser-shooting Robot	45
4.1	Introducción	45
4.2	Diseño esquemático.....	45

4.2.1	Esquemático	47
4.2.2	Listado de componentes	48
4.3	Servomotores.....	49
4.4	Configuración del microcontrolador	50
4.4.1	Configuración de los puertos	50
4.4.2	Configuración TMR0	50
4.4.3	Configuración conversor analógico-digital (CAD)	51
4.4.4	Configuración módulos CCP	51
4.5	Funcionamiento básico	53
4.5.1	Diagramas de flujo.....	54
4.6	Programa fuente	56
4.7	PCB	58
4.8	Diseño 3D	59
4.9	Multimedia	60
5	Cubo de LEDs RGB 3x3x3	61
5.1	Introducción	61
5.2	Diseño esquemático.....	61
5.2.1	LED RGB	61
5.2.2	Esquemático	64
5.2.3	Listado de componentes	65
5.3	Configuración del microcontrolador	66
5.3.1	Configuración de los puertos	66
5.4	Programación	67
5.5	PCB	69
5.6	Montaje.....	71
6	Conclusiones	74
7	Bibliografía.....	75
8	Anexos.....	77
8.1	Anexo I: Hoja de características módulo RTC DS3231	78
8.2	Anexo 2: Partituras de las canciones del Reloj Despertador	98

0 Introducción

El leitmotiv de este trabajo fin de grado es la realización de proyectos electrónicos con microcontroladores PIC que posean un carácter divulgativo y presenten un resultado final tangible.

Este objetivo no significa desistir de realizar proyectos ambiciosos, pero sí que se desea que el trabajo no consista meramente en desarrollar código, ya que habitualmente la comprensión del código desarrollado por otras personas suele resultar larga, complicada y tediosa. Asimismo, se desea que el funcionamiento básico de los proyectos realizados pueda ser entendido de forma sencilla sin que ello implique renunciar a proyectos atractivos.

De este modo, se pretende otorgar al lector las herramientas necesarias para poder reproducir los proyectos electrónicos expuestos en este trabajo, pero otorgándole la libertad necesaria para que pueda introducir las modificaciones que estime oportunas.

Respondiendo a este propósito de divulgación se ha decidido utilizar en los proyectos el microcontrolador PIC16F877A de Microchip. A pesar de que se trata de un microcontrolador que se encuentra en proceso de ser sustituido por otros (así lo indica Microchip en su página web), se ha seleccionado porque todavía es ampliamente utilizado, su configuración es sencilla y sus características concuerdan con el objetivo del trabajo fin de grado.

En cualquier caso, incluso si el microcontrolador empleado dejase de fabricarse de forma inmediata ello no supone que los proyectos desarrollados pierdan su sentido. Utilizar otro microcontrolador implicaría tener que modificar aspectos relativos a la configuración del mismo en la programación pero no afectaría de forma relevante al núcleo del trabajo realizado.

Por último, y una vez más con carácter divulgativo, se ha procurado que los proyectos desarrollados cubran diferentes aspectos de la electrónica, desde los conceptos más básicos (entradas y salidas digitales, conversión analógica-digital, pulsadores) hasta otros más complejos (empleo de funciones anidadas, señales PWM, módulos CCP, pantalla LCD, comunicación I²C,...).

1 Método de trabajo y programas utilizados

Se enumeran en este apartado las fases de las que ha constado la realización de cada uno de los proyectos, desde la idea inicial hasta la implementación final, mencionando los programas utilizados en el diseño y programación del proyecto.

- **Propuesta:** antes de iniciar cada uno de los 3 proyectos, se han definido qué módulos del microcontrolador y qué periféricos constituían la base del mismo, con el objetivo de evitar repetir lo realizado en el proyecto anterior.
 - *Reloj despertador:* sus aspectos más importantes son la comunicación I²C entre el microcontrolador y el módulo DS3231, y el manejo de la pantalla LCD. No obstante, también sobresalen la gestión de interrupciones, la utilización de funciones y el empleo de un zumbador (módulo PWM para generar las notas).
 - *Laser-shooting robot:* proyecto de carácter más práctico en el que se emplean dos servomotores (requieren cada uno una señal PWM, generada con los módulos de comparación), un láser (salida digital) y un joystick (dos entradas analógicas y una entrada digital).
 - *Cubo de LEDs RGB:* se trata del proyecto más sencillo desde un punto de vista meramente electrónico (exclusivamente salidas digitales). No obstante, es un proyecto muy vistoso que también requiere habilidad a la hora de soldar (actividad que no se suele asociar a la electrónica pero imprescindible) y gestionar de forma eficiente un gran número de salidas, ya que en caso contrario la programación puede ser difícil de abordar.
- **Fase de pruebas y programación:** se utiliza la placa PICDEM 2 Plus de Microchip junto con el debugger MPLAB® ICD3 para ir verificando y corrigiendo la programación que se realiza en lenguaje C con MPLAB® IDE v8.89 y el compilador HI-TECH ANSI C

Se comienza realizando programas sencillos para testear el funcionamiento de cada uno de los periféricos de forma independiente y una vez dominado su manejo se va aumentando la complejidad del programa hasta llegar al resultado deseado.

No obstante, después es conveniente optimizar la programación en la medida de lo posible, por ejemplo, mediante el empleo de funciones, si es que no se ha realizado previamente.

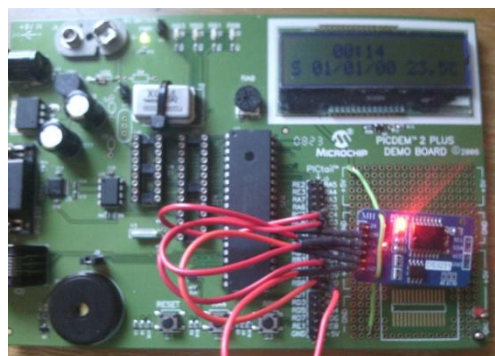


Figura 1. Montaje del reloj-despertador (1^{er} proyecto) en la placa PICDEM 2 PLUS.

- **Montaje en placa:** se efectúa el montaje en una protoboard, empleando todos los componentes y cableando todas las conexiones. Esta etapa permite comprobar que se conocen todas las conexiones requeridas por los periféricos (en la placa PICDEM2 algunos de los periféricos utilizados como la pantalla LCD o el zumbador ya se encuentran conectados) y por el microcontrolador (por ejemplo, la

conexión con el oscilador), así como evitar el olvido de componentes que tienden a ser obviados, como las resistencias de pull-up.

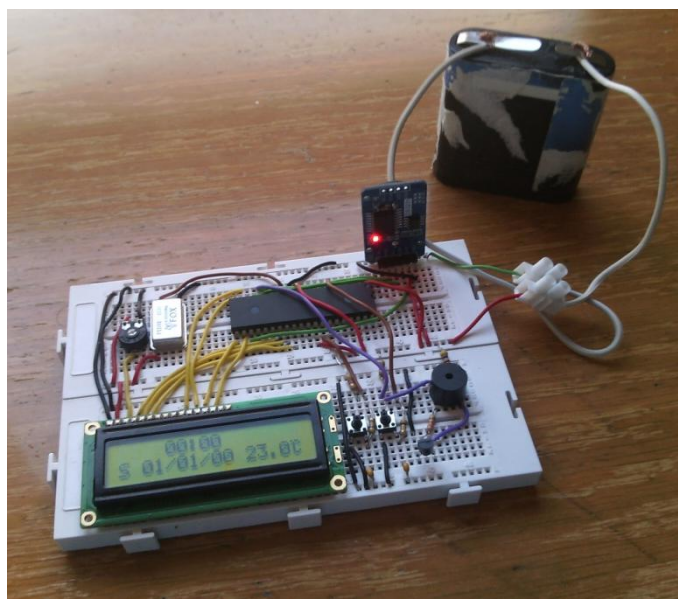


Figura 2 Montaje del reloj- despertador (1^{er} proyecto) en una protoboard.

- **Diseño del esquemático y del PCB:** para diseñar el esquemático y el circuito impreso (PCB, Printed Circuit Board) se ha empleado el programa Designspark PCB 8.0 de la empresa RS Components. Este diseño ya se empieza a realizar en las etapas anteriores, si bien no se da por definitivo hasta concluir el montaje del proyecto en la protoboard.

A la hora de llevar a cabo un diseño se deben considerar las restricciones del fabricante, en este caso, la propia Universidad Pública de Navarra (UPNa), donde se han fabricado los PCB diseñados. Las dimensiones mínimas para la realización de los PCBs en el taller de la UPNa se recogen en la siguiente tabla.






		mm	thou
Ancho de las pistas		0.3	12
Separación entre pistas o pads		0.3	12
Diámetro mínimo del taladro		0.6	24
Pared mínima de la corona		0.25	10
Separación mínima entre pistas y canto (para el fresado)		0.25	10

Tabla 1. Tecnología de fabricación UPNa.

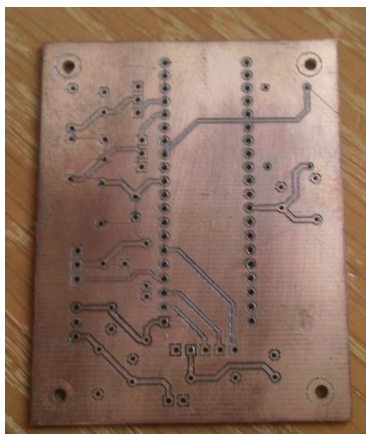


Figura 3. PCB de una capa fabricado con tecnología UPNa.

Se han respetado los límites establecidos en la tabla con el mayor margen posible. En el caso del ancho de las pistas o la separación entre ellas, ello contribuye además a la integridad de la señal. De este modo, no se han empleado en ningún momento anchos de pista inferiores a 0.40 mm y la distancia mínima entre dos elementos cualesquiera (pista, pad, componente, borde) ha sido de al menos 0.30 mm.

Además de estos límites, hay que tener en cuenta que el PCB ha de ser de 1 o 2 capas y que no se dispone de la posibilidad de utilizar vías metalizadas ni serigrafía. Cuando se requieran vías, se utilizará un cable de cobre lo más corto posible y se soldará por ambas caras del PCB.

- **Montaje de los componentes en el PCB:** una vez fabricada el PCB con una fresadora CNC (control numérico por ordenador) se procede al montaje de los componentes utilizando una estación de soldadura de estaño. Sin embargo, previamente es recomendable adoptar una serie de medidas para facilitar el proceso de soldadura.

En primer lugar se comprueba que ninguna de la pistas esté cortada o cortocircuitada con alguno de los planos (masa o alimentación) y que todos los agujeros pasantes (pads o vías) atraviesan la placa. Si es necesario eliminar cobre se recurrirá al uso de una pequeña fresadora portátil.

Posteriormente, se limpia el PCB con alcohol. A continuación, se empiezan a soldar los componentes. Se aplica un lápiz de flux a los pads antes de soldar para favorecer la fluidez del estaño.

Cada vez que se suelda un componente o simplemente un pad se debe comprobar con el polímetro que la soldadura es correcta, es decir, hay continuidad y no se ha cortocircuitado el terminal con alimentación o masa. En caso contrario habrá que volver a soldar el pad, lo que puede requerir la utilización de un desoldador si hay que retirar un exceso de estaño.

(*) *Nota:* en el caso del cubo de LEDs no se ha seguido el orden expuesto. Dado que en este proyecto el montaje de los componentes en el PCB es la parte más compleja, esta se efectúa en primer lugar, y después se lleva a cabo la programación. No se realiza el montaje en placa completo, tan sólo el montaje de un LED para comprobar cuáles son sus conexiones.



Figura 4. Estación de soldadura empleada.

- **Pruebas de funcionamiento:** tras acabar el montaje de los componentes es necesario testear el funcionamiento del circuito. El microcontrolador se programa con el código definitivo en la placa PICDEM 2 PLUS y después se introduce en el zócalo con el que cuenta el PCB.

Es habitual que aparezcan varios fallos antes de lograr el funcionamiento correcto. Los primeros elementos que conviene comprobar son el bloque de alimentación y el oscilador. La forma más sencilla de verificar lo primero es medir la tensión en los planos de masa y de alimentación. Respecto al oscilador, debe medirse la señal que este genera y si llega al microcontrolador, ya que si esto no ocurre, el microcontrolador no puede ejecutar las instrucciones programadas.

Para efectuar estas y otras comprobaciones se requiere un polímetro (continuidad, tensiones), un osciloscopio (permite verificar que las formas de onda en los terminales coinciden con lo esperado) y paciencia.

- **Diseño de carcasa:** salvo en el caso del último proyecto, se ha diseñado una carcasa donde introducir el PCB. Su finalidad es proteger las partes más sensibles del proyecto así como mejorar su apariencia estética.

Se ha empleado el programa de diseño 3D DesignSpark Mechanical 2.0 de la empresa RS Components. En el diseño se han tenido en cuenta aspectos como la alimentación del dispositivo, la posibilidad de fijar el PCB o las partes que deben quedar accesibles para el usuario final.

Las piezas han sido realizadas con la impresora 3D BQ Witbox, propiedad de la Universidad Pública de Navarra (UPNa).

2 Aspectos comunes a todos los proyectos

Se exponen en este apartado algunas características comunes al diseño de los 3 proyectos. El objetivo es evitar su repetición a la hora de explicar cada uno de ellos.

2.1 Diseño del esquemático y componentes utilizados

Los principales aspectos comunes comprenden los requisitos exigidos por el microcontrolador, el bloque de alimentación del circuito y los condensadores utilizados para mejorar tanto el sistema de distribución de la alimentación como la integridad de la señal.

- *Microcontrolador:* se recuerda que el microcontrolador empleado en todos los proyectos es el microcontrolador PIC16F877A. En particular se ha utilizado el modelo de 40 pines con encapsulado PDIP.

Para su funcionamiento se precisa un circuito de alimentación de 5 V, un oscilador (carece de oscilador interno) y que el terminal MCLR se encuentre en alto durante la ejecución del programa (si se encuentra a 0 V se produce un reset).



Figura 5. Microcontrolador PIC16F877A de Microchip, encapsulado PDIP de 40 pines.

En todos los casos se ha usado un oscilador de 4 MHz y se ha conectado el pin MCLR directamente a 5 V. Notar que esta conexión no coincide con la recomendada en la hoja de características del microcontrolador porque no es correcta para la programación del mismo. No obstante, puesto que la programación del microcontrolador se efectúa en la placa PICDEM 2 PLUS, en este caso sí que es válida.

- *Bloque de alimentación:* el punto de partida en los 3 proyectos es una fuente de alimentación de 9 V, ya sea una batería o un transformador que obtiene 9 V a partir de la red eléctrica (o incluso ambas opciones). En el primer caso, se requiere un conector para batería y en el segundo, un conector DC hembra.

A continuación se utiliza un regulador para obtener los 5 V que necesita el microcontrolador. Se ha recurrido a un regulador de conmutación que proporciona 5 V y un máximo de 0.5 A a su salida (salvo en el caso del cubo de LEDs, ver explicación en el 3^{er} proyecto).

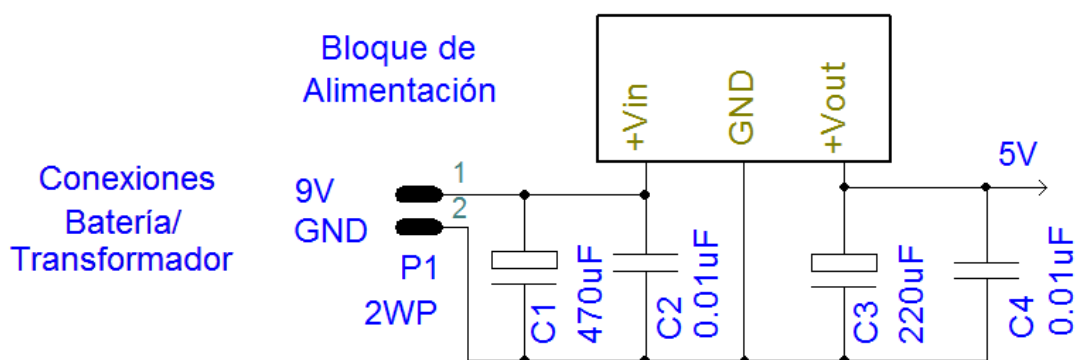


Figura 6. Esquemático del bloque de alimentación.

Para garantizar la estabilidad de la alimentación se coloca un condensador C₁ de 470 µF a la entrada del regulador y un condensador C₃ de 220 µF a la salida del mismo. Los condensadores C₂ y C₄ (0.01 µF) son condensadores de desacoplo.

- *Condensadores de bulk y desacoplo:* en el diseño se incluye un condensador de reserva (bulk) y uno de desacoplo junto a los terminales de alimentación de cada uno de los periféricos y del microcontrolador (este cuenta con dos pares).

Los condensadores de reserva (con capacidad de 47 µF en el caso de los periféricos y el microcontrolador) sirven para mantener estable la tensión de alimentación mientras que los condensadores de desacoplo (0.01 µF) tienen como finalidad suministrar las corrientes transitorias y minimizar el ruido inyectado al sistema de alimentación.

La excepción la constituyen los servomotores, en los que no se han utilizado condensadores de desacoplo (que filtran las altas frecuencias) para evitar la distorsión de las señales PWM generadas con los módulos de comparación. No obstante, no es probable que se hubiera apreciado su efecto (son señales de baja frecuencia, 50 Hz).

2.2 Diseño del PCB

En el diseño de los PCBs se han fijado una serie de criterios comunes, algunos de los cuales derivan de la tecnología de fabricación empleada (ver **Tabla 1**).

- *Componentes*: todos los componentes empleados son de agujero pasante (through-hole) por comodidad a la hora de efectuar la soldadura, más sencilla que en el caso de componentes SMD.
- *Pistas*: se ha mencionado previamente que el ancho mínimo adoptado para las pistas del PCB ha sido de 0.40 mm. En concreto, se han empleado los siguientes anchos de pista:

Tipo de pista	Ancho (mm)
Pista de GND	1.00
Pista de Alimentación (V_{CC})	0.70
Pista de Alimentación (V_{CC}), tramo estrecho	0.50
Pista de Señal	0.40

Tabla 2. Anchos de pista utilizados.

En realidad, en ningún momento ha sido necesario usar pistas de GND, ya que se disponía de plano de masa. La creación del tipo de pista “Pista de alimentación, tramo estrecho”, se debe a que el ancho estándar de las pistas de alimentación (0.70 mm) resulta excesivo para los pads, lo que obliga a reducir el ancho de la pista a 0.50 mm en el tramo de la pista más cercano al pad.

Por otro lado, se ha procurado minimizar la longitud de las pistas y se han utilizado chaflanes en su trazado (*Miter Segment Mode*), evitando los ángulos agudos. Asimismo, en el caso de los PCBs a 2 capas, se ha procurado que las pistas de la capa superior sean ortogonales a las pistas de la capa inferior.

- *Espaciado*: recordar que no se han utilizado espaciados inferiores a 0.30 mm (la restricción se situaba en 0.25 mm). En particular, la separación con los bordes y las formas dibujadas en el cobre se ha establecido en 0.50 mm.
- *Nº de caras del PCB*: el primer proyecto y el último han requerido PCBs a 2 caras. En cualquier caso, se ha restringido el trazado de pistas por la capa superior a los casos en los que era estrictamente necesario, bien porque no era posible trazar las pistas por la cara inferior, bien porque daba lugar a un diseño inadecuado (pistas muy largas, conductores muy juntos,...). Además, no se cuenta con la posibilidad de realizar vías metalizadas, por lo que resulta conveniente restringir su uso al mínimo imprescindible.

El segundo proyecto utiliza un PCB a una cara, fundamentalmente porque este PCB tiene un número muy inferior de pistas al que poseen los otros 2 PCBs.

- *Planos de alimentación (5 V) y masa*: estrictamente hablando sólo se ha utilizado un plano en los 3 proyectos (plano de alimentación en la cara superior del PCB del 2º proyecto).

Para el diseño realizado, es más correcto emplear el término “relleno de tierra” (*“poured ground”*) consistente en rellenar con cobre (conectado a masa) los huecos dejados por las pistas de señal, supliendo la falta de plano de tierra. Una estrategia análoga se puede seguir con la otra cara del PCB y el plano de alimentación.

En los PCBs diseñados, las áreas vacías de la cara inferior se han rellenado con cobre conectado a masa (“plano” de masa) mientras que las áreas vacías de la cara superior se han rellenado con cobre conectado a 5 V (“plano” de alimentación).

- *Acceso a 5 V*: los terminales que se deben conectar a masa se pueden soldar directamente al plano de masa situado en la cara inferior del PCB. No ocurre lo mismo con los terminales que han de conectarse a 5 V, ya que el plano de V_{CC} se sitúa en la cara superior y no todos los componentes pueden soldarse por arriba.

Este problema se produce en el caso de zócalos y condensadores de reserva (electrolíticos). Para solucionarlo, se traza una pista que conecte el terminal correspondiente al pad más cercano conectado a 5 V que se pueda soldar por la cara superior (terminal de un condensador cerámico o una resistencia). Si no es posible, habrá que recurrir a una vía.

- *Empleo de zócalos*: se han utilizado zócalos para el microcontrolador y para el oscilador. En el caso del microcontrolador, para tener la posibilidad de sacarlo y volverlo a programar en la placa PICDEM 2 PLUS (aunque se debería evitar, ya que cuantas más veces se repita esta operación mayor es el riesgo de que se rompa una de la patillas, inutilizando el microcontrolador).

Respecto al oscilador, se emplea un zócalo porque se trata de un componente con patillas endebles y para impedir el cortocircuito entre la carcasa del oscilador (conectada a masa) y la capa superior del PCB (5 V).

- *Agujeros para fijación del PCB (Mounting Holes)*: los PCBs diseñados cuentan con 4 agujeros (uno en cada esquina) que posibilitan su fijación. Poseen un diámetro de 2.30 mm y están conectados a tierra para evitar que la carcasa a la que se fija el PCB pase a estar a un nivel de tensión no nulo.

En la cara superior se encuentran rodeados por un área en forma de corona circular sin cobre para impedir que se produzcan cortocircuitos al introducir los tornillos.

3 Reloj Despertador

3.1 Introducción

Un reloj despertador es un reloj convencional que incluye al menos una alarma que puede ser programada para funcionar como despertador. Existen gran variedad de modelos en el mercado, desde los que sólo indican la hora (habitualmente analógicos) hasta aquellos que también informan sobre la temperatura y la humedad relativa (normalmente digitales).

El objetivo de este proyecto es programar un reloj despertador mediante el empleo del módulo RTC (real time clock) DS3231 y su visualización en un display LCD. El reloj incluirá, además de la hora, calendario (día, mes y año) y temperatura.

La alarma, que se podrá configurar para sonar a una cierta hora del día, cuenta con 3 canciones y también posee modo SNOOZE, que permite postergar 5 minutos la hora de levantarse. Se limitará a 2 el número de veces que se puede usar esta opción.

3.2 Diseño esquemático

En este apartado se indican los componentes (resistencias, condensadores) y las conexiones requeridos por cada una de las partes funcionales en las que se puede subdividir el proyecto. En el caso del bloque de alimentación y el microcontrolador, remitirse al apartado 2.1. Las 2 opciones de alimentación se encuentran disponibles en este proyecto.

Los principales bloques que se pueden distinguir en el proyecto son:

- **Módulo DS3231:** real time clock (RTC), es decir, circuito integrado que almacena el valor de la hora con un alto grado de precisión incluso en ausencia de la fuente de alimentación principal (en este caso utiliza su batería auxiliar).

El microcontrolador “desconoce” la hora y la pregunta continuamente al DS3231 mediante comunicación I²C (inter-integrated circuit), que le responde del mismo modo. Posteriormente se explicará este protocolo en mayor profundidad.

El módulo DS3231 posee 6 terminales: 2 de alimentación ($V_{CC} = 5\text{ V}$ y GND), 2 para la comunicación I²C (SCL y SDA , conectados respectivamente a $RC3$ y $RC4$, pines SCL y SDA del microcontrolador), el terminal SQW (conectado a $RC1$, entrada del módulo CCP2; genera un pulso al saltar la alarma) y el terminal $32K$ (onda cuadrada de 32 kHz, pin no utilizado).

- **Display LCD:** utilizada para la visualización de la hora, se configura en modo de 4 bits (se emplea sólo el subconjunto $D4-D7$ de los pines $D0-D7$ de la pantalla) con dos líneas de 16 caracteres de 5 x 8 puntos.



Figura 7. Módulo DS3231, vista superior e inferior.

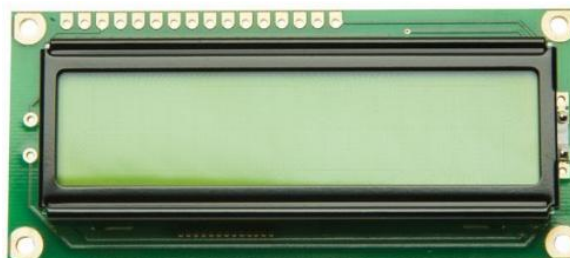


Figura 8. Display LCD 2x16

Las conexiones requeridas por el display LCD son las siguientes:

Función	Nº Pin	Nombre	Conectado al pin/net	Descripción
GND	1	V _{SS}	GND	Conexión Display a masa (0 V)
V _{CC}	2	V _{DD}	5 V	Conexión Display a Alimentación (5 V)
Contraste	3	V _O	Se necesita conectar un potenciómetro (alimentado entre 0 y 5 V) para ajustar el contraste. Una vez calibrado correctamente, no se modificará.	
Control de operación	4	RS	RD4	Selección de registro
	5	R/W	RD5	Lectura/Escritura
	6	E	RD6	Habilitación Display LCD
Datos/comandos	7-10	D0-4	RD0-3	Terminales utilizados en modo 4 bits.
	11-14	D5-8	GND	Terminales no utilizados en modo 4 bits, se conectan a GND.
LED	15,16	A, K	GND	Ánodo, cátodo del LED de iluminación de fondo. No utilizados, se conectan a GND.

Tabla 3. Conexiones display LCD.

(*) **Nota:** en función del modelo de display utilizado, este contará con serigrafía identificando todos los pines o sólo los pines extremos (1 y 16). Una de las causas por las que el display puede “no funcionar” es haber invertido el orden de conexión.

En cualquier caso, al menos en los modelos con los que se ha trabajado, el pin 1 es el situado en la esquina superior izquierda, según la orientación mostrada en la **Figura 8**.

- **2 Pulsadores (Switches):** el pulsador 1 se encuentra conectado a la entrada *RB0* y el pulsador 2 a la entrada *RA4*, ambos a través de una resistencia de 470 Ω . Cada pulsador cuenta además con una resistencia de pull-up (4.7 k Ω) y un condensador de 0.1 μ F, este último para evitar el problema de los rebotes.



Figura 9. Pulsador (de montaje en PCB).

- **Zumbador (Buzzer):** utilizado para generar las notas de las canciones de la alarma, se conecta al pin *RC2*, salida del módulo CCP1. Se requiere un transistor NPN para cortar la corriente del circuito del zumbador, con una resistencia de base (2.2 k Ω) y una de colector (470 Ω).



Figura 10. Zumbador piezoeléctrico.

3.2.2 Listado de componentes

Se incluye a continuación una lista con todos los componentes utilizados en el proyecto. Para los elementos que forman parte del PCB se indica su referencia, una breve descripción, el tipo de empaquetado y su cantidad (parámetro relevante en el caso de resistencias y condensadores). Los elementos que carecen de referencia son aquellos que no se encuentran en el PCB (alimentación y periféricos).

Referencia	Descripción	Valor	Empaquetado	Uds.
P1	Conector alimentación (2 terminales)		DSC	1
P2	Conector RTC DS3231 (6 terminales)		DSC	1
P3	Conector pantalla LCD (16 terminales)		DSC	1
R1, R3	R pull-up (S1-2)	4.7 k Ω	DSC	2
R2, R4	R limitadora de corriente (S1-2)	470 Ω	DSC	2
R5	Resistencia de colector (Q1)	470 Ω	DSC	1
R6	Resistencia de base (Q1)	2.2 k Ω	DSC	1
R7	Potenciómetro	10 k Ω	DSC	1
C1	C _{IN} regulador	470 μ F	DSCV	1
C3	C _{OUT} regulador	220 μ F	DSCV	1
C2, C4-5, C7, C10-11	Condensador de desacoplo	0.01 μ F	DSC	6
C6, C8-9, C12	Condensador de reserva	47 μ F	DSCV	4
C13-14	Condensador anti-rebotes (S1-2)	0.1 μ F	DSC	2
Q1	Transistor bipolar NPN BC549		TO-92	1
LS1	Zumbador piezoeléctrico		DSC	1
S1-2	Pulsador		DISC	2
U1	Microcontrolador PIC16F877A		PDIP	1
U2	Regulador de conmutación R78E5.0-0.5		SIP3	1
Y1	Oscilador f = 4MHz		DIP	1
-	Batería 9 V			1
-	Conector batería			1
-	Transformador 9 V			1
-	Conector DC hembra			1
-	Módulo RTC DS3231			1
-	Display LCD 2 x 16			1

Tabla 4. Listado de componentes proyecto Reloj Despertador.

3.3 Módulo DS3231

En esta sección se explica cómo se configura el módulo DS3231 y cómo se recuperan y modifican los valores del reloj y de la alarma, así como de la temperatura.

Con este propósito, se han dividido los registros del módulo DS3231 en las siguientes categorías: registros de configuración, registros tipo BCD, registros tipo hora, registros de las alarmas y registros de temperatura. Se trata de una clasificación arbitraria, realizada para facilitar la comprensión del programa.

Finalmente, se detalla cómo se efectúa la comunicación I²C entre el microcontrolador y el módulo DS3231.

Aunque se han utilizado prácticamente la totalidad de los registros y opciones del módulo DS3231, ciertos aspectos concretos de su funcionamiento no se recogen en este proyecto. Para ellos, se recomienda consultar las hojas de características del módulo DS3231 (ver anexos), ya que no se explican en este apartado.

3.3.1 Registros de configuración

Existen dos registros de configuración: 0x0E (Registro de Control) y 0x0F (Registro Status). Sólo se remarcan y explican los bits utilizados. Para los demás, consultar la hoja de características.

Registro	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x0E	$\overline{\text{EOSC}}$	BBSQ	CONV	RS2	RS1	INTCN	A2IE	A1IE
0x0F	OSF	0	0	0	EN32kHz	BSY	A2F	A1F

Tabla 5. Registros de configuración DS3231.

- **INTCN**: cuando está a 0, genera una señal cuadrada de frecuencia determinada por RS2 y RS1 en el pin *SQW*. Cuando está a 1, el pin *SQW* se encuentra a 0 en funcionamiento normal y se genera un pulso cuando salta alguna de las alarmas y su correspondiente interrupción está habilitada. En el programa, INTCN se encontrará siempre a 1.
- **A2IE, A1IE**: bits de habilitación de las interrupciones de las alarmas 2 y 1, respectivamente. En función del modo en el que se encuentre el despertador uno o ninguno de los bits se pondrán a 1 (ver explicación módulo CCP2).
- **EN32kHz**: puesto a 1, este bit proporciona una señal de reloj de 32 kHz (frecuencia del oscilador interno del módulo DS3231) en el pin 32K. Como no se requiere esta señal, se va a fijar este bit a 0.
No es una frecuencia excesivamente alta, pero de este modo se evita introducir interferencias adicionales en el resto del circuito.
- **A2F, A1F**: flags (señalizadores) de que ha saltado la alarma 1 o la alarma 2 respectivamente. Estos bits sólo se pueden poner a 0 (no se puede forzar su valor a 1). Cuando ha saltado la alarma, es necesario bajar el flag correspondiente si se desea que la próxima vez que salte la alarma se genere un pulso en el pin *SQW*.

3.3.2 Registros tipo BCD

Se han denominado así los registros de este grupo (ver **Tabla 7**) porque los valores que contienen se encuentran en formato BCD y salvo excepciones su estructura es la que se indica en la siguiente tabla:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	Decenas			Unidades			

Tabla 6. Estructura registros tipo BCD DS3231

El valor almacenado en el bit más significativo (bit 7) varía (ver de nuevo **Tabla 7**). En los registros asociados al reloj este bit suele carecer de utilidad mientras que en los registros asociados a las alarmas suele emplearse en la configuración de la alarma (bits con nombre AxMx), aunque existen 2 casos particulares:

- **Mes (registro 0x05):** el bit 7 es un flag que se activa cuando los años pasan del valor 0 al 99 (cambio de siglo). Como el reloj “sólo” va a programarse para el período 2000-2099 el valor de este bit resulta indiferente, aunque la programación va a hacer que su valor siempre sea 0.
- **Año (registro 0x06):** el bit 7, a diferencia de lo que ocurre en los restantes casos, sí que forma parte de las decenas. Esto supone realizar una distinción en la programación para los años.

Observar que para el año las decenas pueden llegar hasta 9, mientras que para los minutos y los segundos sólo alcanzan el valor de 5.

Se especificará la función de los bits AxMx en los registros vinculados a la alarma. Por el momento, basta indicar que el valor de A1M1, A1M2 y A2M2 es 0 y que este permanece inalterado cuando se modifican los registros tipo BCD.

Nótese el uso de una línea más gruesa en la siguiente tabla para separar registros que ocupan posiciones no consecutivas.

Función	Registro	Bit 7
Segundos (*)	0x00	0
Minutos	0x01	0
Fecha	0x04	0
Mes/Siglo	0x05	Siglo
Año	0x06	0/1
Segundos Alarma 1 (*)	0x07	A1M1
Minutos Alarma 1	0x08	A1M2
Minutos Alarma 2	0x0B	A2M2

Tabla 7. Registros tipo BCD DS3231 y función del bit 7.

A continuación se recoge de forma esquemática la función *act()*, que permite consultar los valores de los registros para actualizarlos en la pantalla LCD o modificarlos porque el usuario desea ajustarlos a un nuevo valor (y al mismo tiempo actualizarlos en la pantalla LCD).

Esta función no se va a emplear para los registros de la **Tabla 7** marcados con (*). Es el caso de los segundos del reloj (registro 0x00), ya que no se va a permitir al usuario alterar este registro ni se va a mostrar en la pantalla LCD; y de los segundos de la alarma (registro 0x07). En este último caso se va a fijar su valor inicialmente a 0x00 y no se modificará durante la ejecución del programa.

Función *act()* (actualizar)

– Variables de entrada:

- *reg* (unsigned char): dirección del registro que se desea actualizar.
- *v0* (unsigned char): valor mínimo del registro (ej: 0 para los minutos).
- *vmax* (unsigned char): valor máximo del registro (ej: 59 para los minutos).
- *x* (unsigned char): variable que indica actualización/incremento (0/1).

– Variable de salida

- *valor* (unsigned char): valor final almacenado en el registro *reg* (*).

– Instrucciones:

- 1) Lectura del registro *reg*.
- 2) Obtención de las decenas (bits 6-4, excepto para los años, bits 7-4) y de las unidades y cálculo del valor decimal ($10 \cdot \text{decenas} + \text{unidades}$).
- 3) Caso $x = 1$, se incrementa el valor.
 - 3.1) Se incrementa el valor en 1, salvo que el valor sea igual a *vmax*. En ese caso se reinicia el valor a *v0*.
 - 3.2) Se calculan las decenas y las unidades a partir del nuevo valor.
 - 3.3) Se calcula el nuevo valor en formato BCD ($16 \cdot \text{decenas} + \text{unidades}$).
 - 3.4) Se escribe en el registro *reg* el nuevo valor. Si se están modificando los minutos del reloj (registro 0x01) se ponen los segundos a 0 (se escribe 0x00 en el registro 0x00) para impedir que los minutos puedan volver a cambiar de valor de forma inmediata.
- 4) El valor (incrementado o no) se muestra en la pantalla LCD (salvo en el caso de los minutos de la alarma 2) en su posición correspondiente.

Apreciar que la posición en la pantalla LCD de la fecha, mes y año cambia según el formato utilizado (europeo o americano).

- (*) **Nota:** sólo se requiere que el programa principal conozca *valor* en el caso de la fecha, el mes y el año, ya que el conocimiento de los tres permite detectar combinaciones incoherentes (31 de abril o 29 de febrero de un año no bisiesto, por ejemplo). Si esta situación se produce se corrige haciendo que la fecha pase a valer 1.

3.3.3 Registros tipo hora

Se incluyen en este grupo los 3 registros que almacenan la hora, correspondiente al reloj, a la alarma 1 y a la alarma 2 respectivamente. En la siguiente tabla se recoge su estructura.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	12 / $\overline{24}$	$\overline{\text{AM}} / \text{PM}$	10 Horas	Unidades de Hora			
		20 Horas					

Tabla 8. Estructura registros tipo hora DS3231.

El significado de los bits 6 y 5 es el siguiente:

- **Bit 6:** indica si el formato es de 12 horas (bit 6 = 1) o de 24 horas (bit 6 = 0).
- **Bit 5:** si el formato es de 12 horas (bit 6 = 1), el bit 5 indica si la hora es AM (bit 5 = 0) o PM (bit 5 = 1). El valor de la hora será $10 \cdot b_4 + b_3b_2b_1b_0$. Si el formato es de 24 horas (bit 6 = 0), el bit 5 sirve para indicar el valor de las decenas de hora junto con el bit 4. El valor de la hora será $10 \cdot (b_5b_4) + b_3b_2b_1b_0$.

Nótese que en el formato de 12 horas el valor de la hora está comprendido entre 1 y 12 (valor máximo de las decenas de hora es 1) y en el formato de 24 h el valor de la hora está comprendido entre 0 y 23 (valor máximo de las decenas de hora de 2).

Nuevamente, la función del bit 7 no es siempre la misma (ver **Tabla 9**). En la hora del reloj este bit carece de utilidad, mientras que en las horas de las alarmas el bit 7 se emplea para la configuración de la alarma.

Se especificará la función de los bits AxMx en los registros vinculados a la alarma. Por el momento, basta indicar que el valor de A1M3 y A2M3 es 0 y que este permanece inalterado cuando se modifican los registros tipo hora.

Función	Registro	Bit 7
Horas	0x02	0
Horas Alarma 1	0x09	A1M3
Horas Alarma 2	0x0C	A2M3

Tabla 9. Registros tipo hora DS3231 y función del bit 7.

Aunque el formato en el que se almacena la hora en realidad también es BCD, no se puede utilizar la función *act()* anteriormente definida porque hay un mayor número de factores a tener en cuenta en este caso.

Se debe distinguir si la hora se encuentra en formato de 12 o de 24 horas y actuar de forma distinta en función de ello. Además, también es interesante que la misma función sea capaz de pasar de formato de 12 a 24 horas si el usuario así lo desea, lo que también incrementa la complejidad.

Todo ello ha llevado a crear una nueva función llamada *act_hora()*, que sirve para consultar el valor de los registros tipo hora, modificar su valor o cambiar el formato de 12/24 horas. Se explica de forma esquemática en la página siguiente.

Función *act_hora()* (actualizar hora)

– **Variables de entrada:**

- *al* (unsigned char): *al* = 0 para la hora del reloj, *al* = 1 para la hora de la alarma 1 y *al* = 2 para la hora de la alarma 2.
- *y* (unsigned char): valor que indica actualización (*y* = 0), incremento (*y* = 1) o cambio de formato 12/24 horas (*y* = 2).

– **Instrucciones:**

- 1) Lectura del registro correspondiente según el valor de *al*.
- 2) Lectura del formato 12/24 horas (bit 6 del registro).
 - 2.1) Formato 24 horas: obtención decenas (bits 5-4) y unidades (bits 3-0).
 - 2.2) Formato 12 horas: obtención decenas (bit 4), unidades (bits 3-0) y hora AM ó PM (bit 5).
- 3) Caso *y* > 0
 - 3.1) Caso *y* = 1, se incrementa el valor de la hora.
 - Se incrementa el valor en 1 salvo que sea igual al valor máximo de la hora (23 para formato 24 h y 12 para formato 12 h). En ese caso se reinicia al valor mínimo (0 para formato 24 h y 1 para formato 12 h).
 - Si el formato es de 12 h y el valor tras el incremento es 12 se cambia la hora de AM a PM o viceversa.
 - 3.2) Caso *y* = 2, se cambia el formato 12/24 horas.
 - Formato 24 horas → Formato 12 horas
 - Hora > 12: se resta 12, la hora es PM.
 - Hora = 0: hora pasa a ser 12 AM.
 - Hora = 12: hora pasa a ser 12 PM.
 - $1 \leq \text{Hora} \leq 11$: valor hora igual, la hora es AM.
 - Formato 12 horas → Formato 24 horas
 - Horas AM: ningún cambio excepto para 12 AM, que pasa a 0.
 - Horas PM: se suma 12 excepto para 12 PM, que se queda igual.
 - 3.3) Se calculan decenas y unidades a partir del nuevo valor de la hora.
 - 3.4) Se calcula el valor a escribir en el registro correspondiente.
 - Formato 24 horas → $64 * 0 + 16 * \text{decenas} + \text{unidades}$
 - Formato 12 horas → $64 * 1 + 32 * \overline{\text{AM/PM}} + 16 * \text{decenas} + \text{unidades}$

Si se ha incrementado (*y* = 1) la hora del reloj (*al* = 0) se ponen los segundos a 0 (se escribe 0x00 en el registro 0x00) para impedir que la hora pueda volver a cambiar de valor de forma inmediata.
- 4) Si se está actuando sobre la hora del reloj (*al* = 0) o se está actualizando o modificando la hora de la alarma 1 (*y* = 0 ó 1, *al* = 1) se muestra el valor correspondiente en la pantalla LCD.

(*) **Nota:** cuando el usuario modifica el formato 12/24 horas de la hora del reloj, se modifica por programa el formato 12/24 horas tanto de la hora del reloj como de las alarmas 1 y 2 para que el formato sea igual en los 3 casos. El módulo DS3231 permite que los formatos no sean coincidentes pero se ha considerado más adecuado garantizar que sea el mismo para las 3 horas.

3.3.4 Registros de las alarmas

Se agrupan bajo esta denominación los registros vinculados a las alarmas. Todos ellos excepto 2 (0x0A y 0x0D) ya se han incluido en los 2 tipos anteriores de registros. Se les dedica un punto de la explicación porque todavía no se ha detallado la función de los bits AxMx. En la tabla inferior aparecen los registros de esta clase y su respectivo bit AxMx, que siempre ocupa el bit 7.

Función	Registro	Bit 7
Segundos Alarma 1	0x07	A1M1
Minutos Alarma 1	0x08	A1M2
Horas Alarma 1	0x09	A1M3
Día/Fecha Alarma 1	0x0A	A1M4
Minutos Alarma 2	0x0B	A2M2
Horas Alarma 2	0x0C	A2M3
Día/Fecha Alarma 2	0x0D	A2M4

Tabla 10. Registros de las alarmas DS3231 y bit AxMx correspondiente.

La función de los bits AxMx es determinar bajo qué condición salta la alarma. En este caso se desea que tanto la alarma 1 como la alarma 2 salten cuando horas y minutos (para la alarma 1, también segundos) del reloj coincidan con la hora programada. Ello se consigue con los siguientes valores de los bits AxMx:

A1M4 = 1, A1M3 = 0, A1M2 = 0, A1M1 = 0

A2M4 = 1, A2M3 = 0, A2M2 = 0 (A2M1 no existe)

Para todos los bits AxMx, excepto para A1M4 y A2M4, ya se ha asegurado previamente que su valor es el adecuado. Para A1M4 y A2M4, simplemente se fija su valor a 1 al inicio del programa.

Los registros 0x0A y 0x0D sirven para fijar qué día de la semana o del mes se desea que salte la alarma (si se exige también esta condición). Como en este caso no se emplean, su valor no se modifica durante la ejecución del programa, luego se puede asegurar que el valor de A1M4 y A2M4 permanecerá igual a 1.

Para saber cómo configurar los bits AxMx para que las alarmas salten bajo otras condiciones distintas, consultar las hojas de características del módulo DS3131.

3.3.5 Registros de temperatura

De los registros utilizados, únicamente faltan por mencionar los 2 registros ligados a la temperatura, mostrados en la siguiente tabla.

Registro	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x11	Signo T ^a	Parte entera Temperatura						
0x12	Parte decimal T ^a		0	0	0	0	0	0

Tabla 11. Registros de temperatura DS3231.

El signo de la temperatura se ignora, ya que se considera que en el interior de una vivienda (donde se pretende utilizar el reloj despertador) no se van a alcanzar temperaturas inferiores a 0°C. El módulo DS3131 proporciona la temperatura en grados Celsius (°C), con la parte entera en binario (no en BCD, como los restantes registros) y 2 bits para la parte decimal, proporcionando una resolución de 0.25 °C.

Como en el display LCD sólo queda espacio para 1 cifra decimal se escribirá en la pantalla LCD “.0” si el valor del registro 0x12 es inferior a 2 y “.5” en caso contrario.

Si el usuario desea visualizar la temperatura en grados Fahrenheit (°F), la conversión de °C a °F se realiza por programa, ya que el módulo DS3231 no ofrece la posibilidad de cambiar de escala. En este caso, la temperatura se muestra sin parte decimal por falta de espacio en el display LCD.

La función que se utiliza para actualizar la temperatura es:

Función *act_Ta()* (actualizar T^a)

– Instrucciones:

- 1) Se escriben las barras (“/”) que separan día, mes y año en la posición que corresponda según el formato (europeo, *fr* = 0; o americano, *fr* = 1).
- 2) Lectura parte entera de T^a, cálculo de decenas (*DT*) y unidades (*UT*) de T^a.
- 3) Cálculo del valor de la parte decimal (*dT*) de T^a (valor entre 0-3).
- 4) Escritura de T^a en display LCD.

4.1) Caso *fr* = 0 (formato europeo, T^a en °C)

- Se escribe *DT UT . 0* si *dT* < 2;
ó *DT UT . 5* si *dT* ≥ 2.
- Escritura del símbolo °C.

4.2) Caso *fr* = 1 (formato americano, T^a en °F)

- Se calcula la T^a en °F como
 $T = 9 \cdot (10 \cdot DT + UT) / 5 + 32$
- Se calculan los nuevos valores de *DT* y *UT*.
- Se escribe *DT UT °F* en el display LCD.

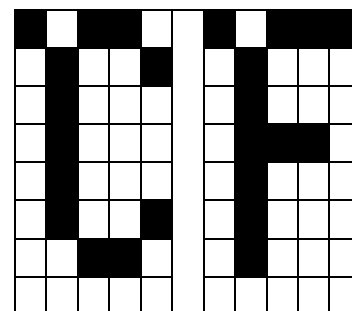


Figura 12. Caracteres *celsius* y *fahrenheit*.

A la derecha se pueden ver los símbolos *celsius* y *fahrenheit*, almacenados en las posiciones 0 y 1 de la memoria CGRAM del display LCD. Se definen mediante la función *charCGRAM()* (ver programa fuente).

3.3.6 Comunicación I²C

El microcontrolador se comunica con el módulo RTC DS3231 mediante el protocolo I²C. Se trata de un protocolo de comunicaciones síncrono (se requiere una señal de reloj) que emplea un bus bidireccional de dos líneas: SDA (serial data), línea utilizada para transmitir los datos; y SCL (serial clock), línea de reloj para sincronizar la transmisión de dichos datos.

La arquitectura es de tipo maestro-esclavo, con el microcontrolador actuando como maestro (gobierna la comunicación, tomando la iniciativa; y genera la señal de reloj) y el módulo DS3231 como esclavo (podría haber más esclavos, aunque no es el caso).

Se va a explicar el protocolo de comunicaciones basándose en imágenes extraídas del datasheet del DS3231 y en las funciones utilizadas en el programa. Estas llaman a su vez a las funciones de la librería I2c.c, incluida en el proyecto de MPLAB.

Se deben distinguir dos tipos de operaciones: escritura de un nuevo valor en un registro y lectura del valor de un registro.

Operación de escritura

Función *escribir()*

– Variables de entrada:

- *ry* (unsigned char): dirección del registro que se desea sobrescribir.
- *dy* (unsigned char): datos que se desean escribir en el registro *ry*.

– Instrucciones:

- 1) Secuencia de **START** enviada por el maestro (función *i2c_start()*).
- 2) Maestro envía al bus la dirección del esclavo (DS3231), con el que desea comunicarse, **0b1101000** (7 bits), más el bit de dirección (R/W), igual a **0** para una orden de escritura. Se utiliza la función *i2c_write(0xD0)*.
- 3) Esclavo envía al maestro un **ACK** (acknowledge) para confirmar la recepción del byte transmitido por el maestro.
- 4) Maestro envía al esclavo la dirección del registro *ry* cuyo valor se desea sobrescribir mediante *i2c_write(ry)* y el esclavo devuelve un **ACK**.
- 5) Maestro envía al esclavo el valor *dy* a escribir en el registro *ry* con la instrucción *i2c_write(dy)* y el esclavo devuelve un **ACK (*)**.
- 6) Secuencia de **STOP** enviada por el maestro (función *i2c_stop()*) para finalizar la operación de escritura.

(*) **Nota:** esta función modifica el valor de un único registro. Para sobrescribir los valores de varios registros consecutivos, tras enviar *dy*, se deberían seguir enviando los valores a escribir en los sucesivos registros.

Después de cada orden de escritura, el valor del puntero se incrementa en 1 (es decir, pasa de *ry* a *ry + 1*, *ry + 2*,...) lo que permite cambiar el valor de varios registros en una sola transmisión. Tras cada byte, el esclavo debe enviar un **ACK**.

La siguiente figura refleja la operación de escritura descrita:

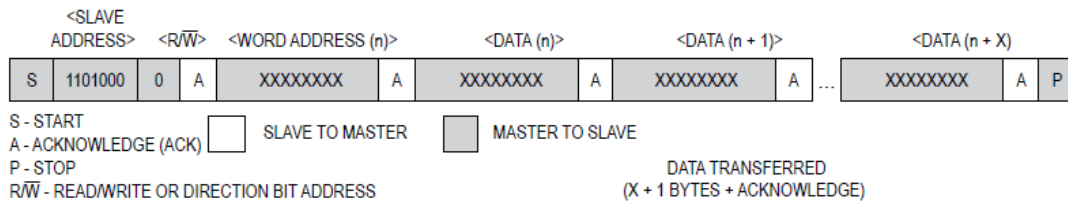


Figura 13. Protocolo de comunicaciones I²C. Operación de escritura.

Operación de lectura

Función *leer()*

- **Variables de entrada:** *rx* (unsigned char): dirección del registro a leer.
- **Variables de salida:** *dx* (unsigned char): datos leídos del registro *rx*.

– Instrucciones:

- 1) Secuencia de **START** enviada por el maestro (función *i2c_start()*).
- 2) Maestro envía la dirección del esclavo seguida del bit R/W, igual a 0 (orden de escritura). Se utiliza la función *i2c_write(0xD0)*.
- 3) Esclavo envía al maestro un **ACK** (acknowledge).
- 4) Maestro envía al esclavo la dirección del registro *rx* cuyo valor se desea leer mediante *i2c_write(rx)* y el esclavo devuelve un **ACK** (*).
- 5) Se vuelve a enviar la secuencia de START (función *i2c_restart()*).
- 6) Maestro vuelve a enviar la dirección del esclavo seguida del bit R/W, pero ahora su valor es **1** (orden de lectura). Se utiliza la función *i2c_write(0xD1)*.
- 7) Esclavo envía al maestro un **ACK** (acknowledge).
- 8) Esclavo envía el contenido del registro *rx* al maestro, que lo guarda en *dx* mediante la instrucción *dx = i2c_read()* (**).
- 9) Maestro envía al esclavo un **NACK** (not acknowledge) mediante la función *i2c_sendnack()* seguido de la secuencia de STOP (función *i2c_stop()*) para finalizar la operación de lectura.

(*) **Nota:** se requiere una operación de escritura en primer lugar para que el puntero apunte al registro que se desea leer.

(**) **Nota:** si el maestro enviase un **ACK** (función no implementada en la librería I2c.c) recibiría el valor del siguiente registro, y así sucesivamente hasta que se decida pasar a 9). No se ha utilizado esta opción.

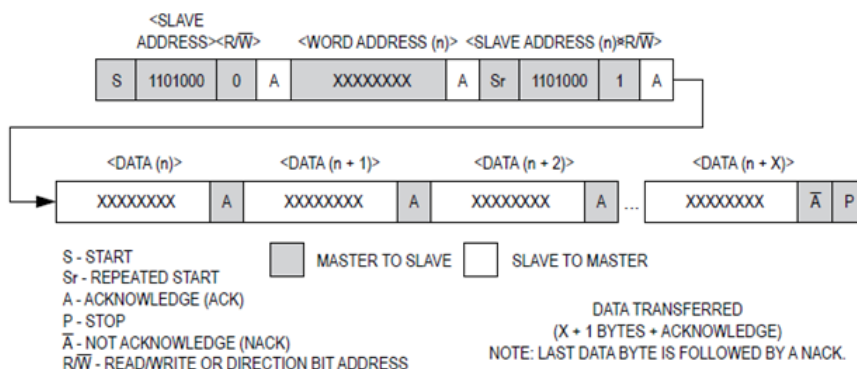


Figura 14. Protocolo I²C. Operación de escritura/lectura (en primer lugar se fija el puntero al valor deseado, después se lee).

3.4 Configuración del microcontrolador

En este apartado se recoge la configuración de los puertos y de los módulos del microcontrolador utilizados en el proyecto. Los módulos que se han empleado son: TMR0, TMR1, CCP1 (junto con TMR2 en modo PWM) y CCP2 (en modo Captura), el puerto SSP (configurado en modo I²C) y el puerto PSP (empleado para el display LCD).

El control del puerto SSP (Synchronous Serial Port) y del puerto paralelo utilizado por el display LCD no se va abordar en detalle, ya que se utilizan funciones de las librerías I2c.c y lcd.c respectivamente para gobernar su funcionamiento.

3.4.1 Configuración de los puertos

Los terminales del microcontrolador utilizados son:

ENTRADAS			SALIDAS	
Pin	A/D	Descripción	Pin	Descripción
<i>RA4</i>	D	Pulsador 1 (SW1)	<i>RC2</i>	Salida Módulo CCP1
<i>RC1</i>	D	Entrada Módulo CCP2	<i>SCL</i>	Serial Clock Comunicación I ² C
<i>RB0</i>	D	Pulsador 2 (SW2)	<i>RD0-3</i>	Pines D4-D7 display LCD
			<i>SDA</i>	Serial Data Comunicación I ² C
			<i>RD4</i>	Pin RS display LCD
			<i>RD5</i>	Pin R_W display LCD
			<i>RD6</i>	Pin E display LCD

Tabla 12. Entradas y salidas del microcontrolador (Reloj Despertador).

Por tanto, los puertos se configuran del siguiente modo:

Registros y comandos

- i2c_init();* ⇒ Se inicializa el protocolo I²C, configurando *SCL* (*RC3*) y *SDA* (*RC4*) como salidas (*TRISC3* = 1; *TRISC4* =1).
- lcd_init();* ⇒ Se inicializa el display LCD, configurando los registros necesarios (entre ellos *PORTD*). Conviene ejecutar esta función antes de configurar el resto de registros.
- ADCON1* = 0x06; ⇒ *PCFG* = 0b0110 ⇒
 ⇒ *RA0-RA3*, *RA5* y *RE0-RE2*; terminales digitales (no es estrictamente necesario porque no se utilizan).
- TRISA* = 0xFF; ⇒ *PORTA*: todos los terminales configurados como entradas.
- TRISB* = 0xFF; ⇒ *PORTB*: todos los terminales configurados como entradas.
- TRISC* = 0xFB; ⇒ *PORTC*: todos los terminales, entradas; salvo *RC2*, salida.

3.4.2 Configuración TMR0

TMR0 se utiliza para detectar pulsos en *RA4* (entrada de flancos para TMR0 en modo contador). Se gestiona por interrupción.

Registros

GIE = 1; T0IE = 1; ⇒ Interrupción global e interrupción TMR0 habilitadas.
 OPTION_REG = 0xBF; ⇒ T0CS = 1 ⇒ TMR0 configurado como contador
 ⇒ T0SE = 1 ⇒ TMR0 se incrementa con flanco ↓ en *RA4*
 ⇒ PSA = 1 ⇒ Predivisor asignado a WDT ⇒
 ⇒ TMR0 sin predivisor
 TMR0 = 255; ⇒ Configuración de TMR0 para que salte la
 interrupción con una pulsación de *RA4*.
 (Cada vez que salta la interrupción hay que volver a poner
 TMR0 = 255, y por supuesto, bajar el flag, T0IF = 0)

Para *RA4* (pulsador 1) es importante considerar dos situaciones de funcionamiento:

- *Pulsación única/mantenida*: situación que se produce cuando hay que ajustar algún parámetro, ya sea del reloj (TIME SET) o de la alarma (ALARM SET). Se utiliza un bucle *while* del que no se sale mientras T0IF = 1 (1ª pulsación) ó *RA4* = 0 (pulsación mantenida)
- *Distinción pulsación corta/larga*: situación que se produce cuando el despertador está sonando y se desea apagarlo, o al menos que pare de forma momentánea. Se emplea un bucle *while* del que no se sale mientras *RA4* = 0 ó un contador auxiliar (*p*, que se incrementa en el interior del bucle) no haya llegado a un valor límite.

Después se evalúa si se ha salido del bucle porque *RA4* = 1 (pulsación corta, inferior a 5 s) o porque el contador *p* ha alcanzado el valor límite (correspondiente a una temporización de en torno a 5 s).

Para *RB0* (pulsador 2) también es necesario diferenciar entre pulsación corta y larga, por lo que se emplea la misma estrategia. No obstante, para apagar la alarma se considera que la pulsación larga de *RB0* es de 5 s, mientras que para cambiar el modo del reloj despertador la pulsación larga es de tan sólo 0.8 s aproximadamente.

3.4.3 Configuración TMR1

TMR1 se configura para temporizar 500 ms. Con la utilización de un contador auxiliar, *j*, de valor máximo 2, se logra temporizar 1s. Se gestiona por interrupción.

Cada vez que transcurra este tiempo se actualiza la información mostrada en el display LCD, incluyendo el modo (LOCK, TIME SET o ALARM SET), el estado de la alarma (OFF, ON o SNOOZE), y la posición del cursor.

Registros

GIE = 1; PEIE = 1; ⇒ Int. global e interrupciones por periféricos habilitadas.
 TMR1IE = 1; ⇒ Interrupción TMR1 habilitada.
 T1CON = 0x30; ⇒ T1CKPS1-0 = 11 ⇒ Predivisor = 8

$T1CON = 0x30;$ $\Rightarrow TMR1CS = 0 \Rightarrow$ Reloj interno ($F_{Osc}/4$)
 $TMR1H=11;$ $TMR1L=220;$ (valores escritos cada vez que se reconfigura TMR1,
 recordar que para encender TMR1, $TMR1ON = 1$)

$$TMR1 = TMR1H:TMR1L = 256 * TMR1H + TMR1L =$$

$$= 256 * 11 + 220 = 3036$$

Temporización TMR1: $Td = (65536 - TMR1) * P * (4/F_{Osc}) =$
 $= (65536 - 3036) * 8 * (4/4MHz) = 500 \text{ ms}$

Otras funciones

Cada vez que transcurre un segundo, además de actualizar la hora del reloj o de la alarma (si se está configurando esta), también se actualizan otros elementos:

- *Modo del reloj:* las dos primeras posiciones del display LCD se reservan para indicar el modo del reloj. Se recogen bajo estas líneas los símbolos utilizados.

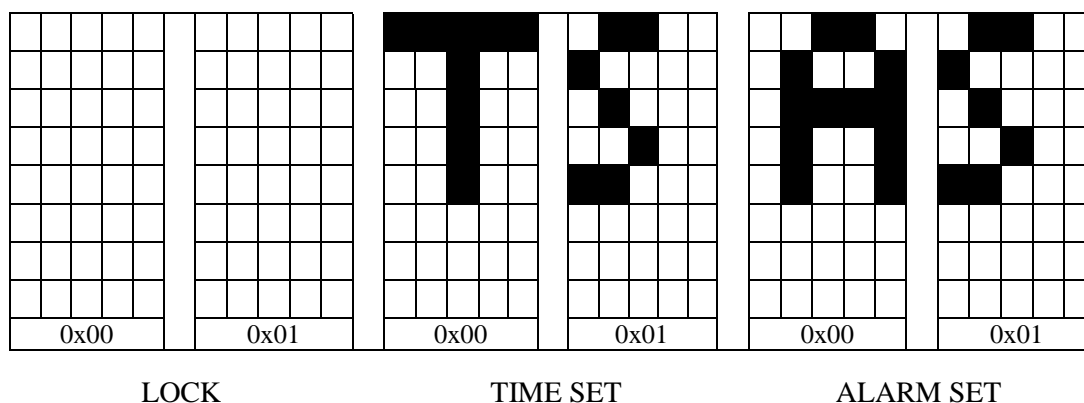


Figura 15. Caracteres mostrados según el modo del reloj.

Estos símbolos, *TIME* (letra 'T' reducida), *ALARM* (letra 'A' reducida) y *SET* (letra 'S' reducida); se encuentran almacenados respectivamente en las posiciones 2, 3 y 4 la memoria CGRAM del display LCD. Se definen mediante la función *charCGRAM()* (ver programa fuente).

- *Modo de la alarma:* las posiciones 0x0D y 0x0E del display LCD se reservan para indicar el modo de la alarma. Los símbolos utilizados son:

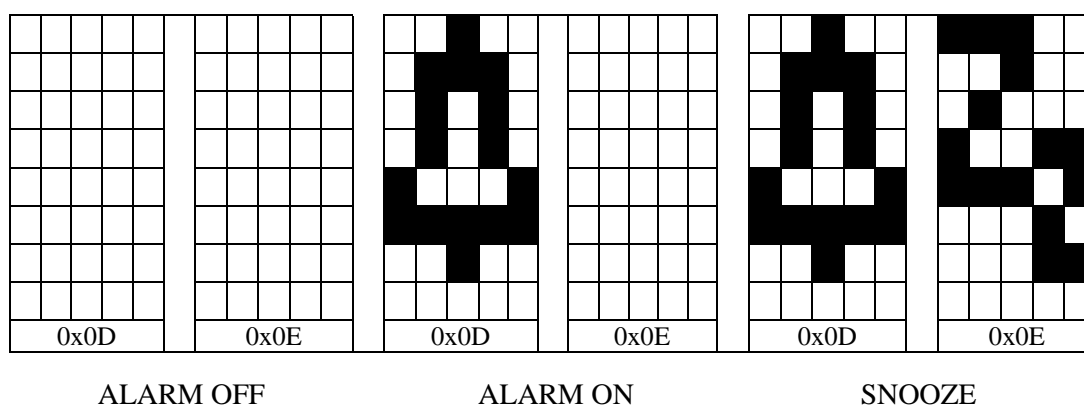


Figura 16. Caracteres mostrados según el modo de la alarma.

Los símbolos creados, *campana* y *snooze* (onomatopeya ‘Zz’) se guardan en las posiciones 5 y 6 de la memoria CGRAM del display LCD. Se crean mediante la función *charCGRAM()* (ver programa fuente).

- **Posición del cursor:** la función *cursor()* sirve para actualizar la posición del cursor (guión que se sitúa en la línea 8 de la posición a la derecha de la última escrita).

Si el reloj está en modo LOCK no se muestra el cursor. Si el reloj se encuentra en modo TIME SET o ALARM SET, sí que se visualiza el cursor, que se coloca siempre en la posición de las unidades del parámetro a cambiar (por ejemplo, las unidades de minutos). También se tiene en cuenta que para algunos parámetros (fecha, mes, año) la posición del cursor es distinta según el formato.

3.4.4 Módulo CCP1

El módulo CCP1, configurado en modo PWM (se usa TMR2), se utiliza para generar las notas musicales del despertador mediante el empleo de la función *nota()*. Para cada nota se debe considerar la frecuencia (o lo que es lo mismo, el período) y su duración. El ciclo de trabajo es siempre del 50%.

Se recogen en primer lugar las tablas que indican los valores a introducir en los registros CCPR1L (que fija TON) y PR2 (que establece el período) y el número que se asigna a cada nota en función de su duración. A continuación se explicará la función *nota()*.

Nota musical	Código	f (Hz)	T (μs)	T/16	PR2 (T/16 – 1)	CCPR1L (T/16·0,5)
Mi alto (MiH)	0x00	659	1517	95	94	48
Re alto (ReH)	0x01	587	1704	107	106	54
Do alto (DoH)	0x02	523	1912	120	119	60
Si	0x03	494	2024	127	126	64
Si b/ La #	0x04	466	2146	134	133	67
La	0x05	440	2273	142	141	71
Sol	0x06	392	2551	159	158	80
Fa	0x07	349	2865	179	178	90
Mi bajo (MiL)	0x08	330	3030	189	188	95
Re bajo (ReL)	0x09	294	3401	213	212	107
Do bajo (DoL)	0x0A	262	3817	239	238	120
Silencio (Sil)	0x0B	-	-	-	250	0

Tabla 13. Valores PR2 y CCPR1L para cada nota.

$$T = (\mathbf{PR2} + 1) * P * 4/F_{OSC} = (\mathbf{PR2} + 1) * 16 * 4/4\text{MHz} = (\mathbf{PR2} + 1) * 16 \mu\text{s}$$

$$\mathbf{M} = 4 * \mathbf{M}_8 + \mathbf{M}_2 = 4 * \mathbf{CCPR1L} + 0 = 4 * \mathbf{CCPR1L}$$

$$\begin{aligned} T_{ON} &= \mathbf{M} * P * 1/F_{OSC} = 4 * \mathbf{CCPR1L} * P * 1/F_{OSC} = \\ &= 4 * \mathbf{CCPR1L} * 16 * 1/4\text{MHz} = \mathbf{CCPR1L} * 16 \mu\text{s} \end{aligned}$$

Duración	Factor
Blanca	0x08
Negra con puntillo	0x06
Negra	0x04
Corchea	0x02

Tabla 14. Factor en función de la duración de la nota.

$$\text{Duración de la nota} = \text{Factor} * 125 \text{ ms}$$

El despertador dispone de 3 canciones distintas para despertarse: 1: “Diana de San Fermín” (“Levántate pamplonica”), 2: “ Las mañanitas” y 3: “Quinto levanta”. Los códigos y los factores de duración de las notas se encuentran almacenados en una estructura (*canc*, de tipo *partitura*) de 6 arrays de 64 elementos. Los arrays pares (0, 2, 4) corresponden al código de las notas de cada canción mientras que los arrays impares son los factores de las notas de cada canción. Consultar los anexos para ver las partituras.

Función *nota()*

– Variables de entrada:

- *u* (unsigned char): código de la nota (ver columna 2, **Tabla 13**).
- *v* (unsigned char): factor de duración (ver **Tabla 14**).

– Instrucciones:

- 1) Módulo CCP1 en modo PWM. **CCP1CON = 0x0C** (con $M_2 = 0$)
- 2) Se asigna un valor a CCPR1L en función de la **Tabla 13** y de *u*.
- 3) Se asigna un valor a PR2 en función de la **Tabla 13** y de *u*.
- 4) Se enciende TMR2 (**TMR2ON = 1**) con predivisor igual a 16 mediante la instrucción **T2CON = 0x06**.
- 5) Se esperan 125 ms (con un *delay*) tantas veces como indica *v*.
- 6) Se apaga TMR2 (**T2CON = 0x00**) y se resetea CCP1 (**CCP1CON = 0x00**).

3.4.5 Módulo CCP2

El módulo CCP2 se utiliza para detectar el pulso que envía el circuito integrado DS3231 cuando salta la alarma. Se gestiona por interrupción.

Registros

- GIE = 1; PEIE = 1; ⇒ Int. global e interrupciones por periféricos habilitadas.
CCP2IE = 1; ⇒ Interrupción módulo CCP2 habilitada.
CCP2CON = 0x04; ⇒ Modo Captura, se captura cada flanco de bajada.

Cuando salta la interrupción de CCP2 porque se ha detectado un flanco se activa el despertador para que la alarma empiece a sonar de forma inmediata.

Es importante distinguir cuál es la alarma del DS3231 que ha saltado, AL1 (utilizada para la hora programada por el usuario) o AL2 (empleada para el modo SNOOZE), ya que se actuará en consecuencia. Si el usuario ha programado el modo SNOOZE, cuando salte la alarma (salvo que sea la tercera vez), habrá que programar AL2 para que salte de nuevo a los 5 minutos.

3.5 Funcionamiento básico

Después de haber explicado la configuración del módulo DS3231 así como la del microcontrolador y sus diferentes módulos y haber desarrollado las diferentes funciones utilizadas, se va a dar una coherencia al conjunto explicando el funcionamiento del reloj despertador de forma comprensible para cualquier usuario, independientemente de su nivel de conocimientos de electrónica.

3.5.1 Cómo configurar el reloj (TIME SET)

El modo TIME SET permite al usuario configurar el reloj y el calendario. Para cambiar de LOCK (modo normal de visualización) a TIME SET, *RB0* (en adelante P1, Pulsador 1) debe mantenerse presionado durante al menos 0.8 s. En ese momento, las letras 'TS' aparecen en la esquina superior izquierda del display LCD.

En TIME SET, pulsar P1 cambia el parámetro seleccionado mientras que pulsar *RA4* (en adelante P2, Pulsador 2) posibilita ajustarlo (mantener P2 pulsado ajustará el parámetro con mayor rapidez). Cuando el usuario haya terminado de configurar el reloj, puede volver a LOCK pulsando P1 hasta que las letras 'TS' desaparezcan (en torno a 0.8 s).

En TIME SET se pueden ajustar los siguientes parámetros:

- **Horas y minutos**
- **Formato 12/24h.**
- **Año** (2000-2099), solo se muestran los 2 últimos dígitos.
- **Mes y fecha.**
- **Formato de la fecha:** europeo (DD/MM/AA) o americano (MM/DD/AA)
- **Temperatura:** °C (formato europeo) o °F (formato americano)
- **Día de la semana:** se actualiza automáticamente al cambiar el año, el mes o la fecha. En el formato europeo se muestran las letras españolas para el día de la semana ('L', 'M', 'X', 'J', 'V', 'S', 'D') mientras que en el americano aparecen las letras anglosajonas ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun').

3.5.2 Cómo configurar la alarma (ALARM SET)

ALARM SET permite al usuario configurar la alarma. Para cambiar de LOCK a ALARM SET, se debe pulsar P1 durante al menos 0.8 s. En ese momento, las letras 'AS' aparecen en la esquina superior izquierda del display LCD.

En ALARM, pulsar P1 cambia el parámetro seleccionado mientras que pulsar P2 posibilita ajustarlo (mantener P2 pulsado ajustará el parámetro con mayor rapidez). Cuando el usuario haya terminado la configuración del reloj, puede volver a LOCK pulsando P1 hasta que las letras 'AS' desaparezcan (en torno a 0.8 s).

En ALARM SET se pueden ajustar los siguientes parámetros:

- **Horas y minutos**
- **Modo de la alarma:** OFF, ON o SNOOZE. Aparecerá el correspondiente símbolo, que se mantendrá tanto en LOCK como en TIME SET
- **Canción:** se puede elegir una de las siguientes canciones: 1: "Diana de San Fermín"; 2: "Las mañanitas"; 3: "Quinto levanta".

Se muestra a continuación un diagrama, basado en una máquina de estados finitos, que procura sintetizar de forma gráfica la configuración del reloj (TIME SET) y de la alarma (ALARM SET). Observar la leyenda situada en la parte inferior de la imagen, que indica el significado de cada uno de los símbolos.

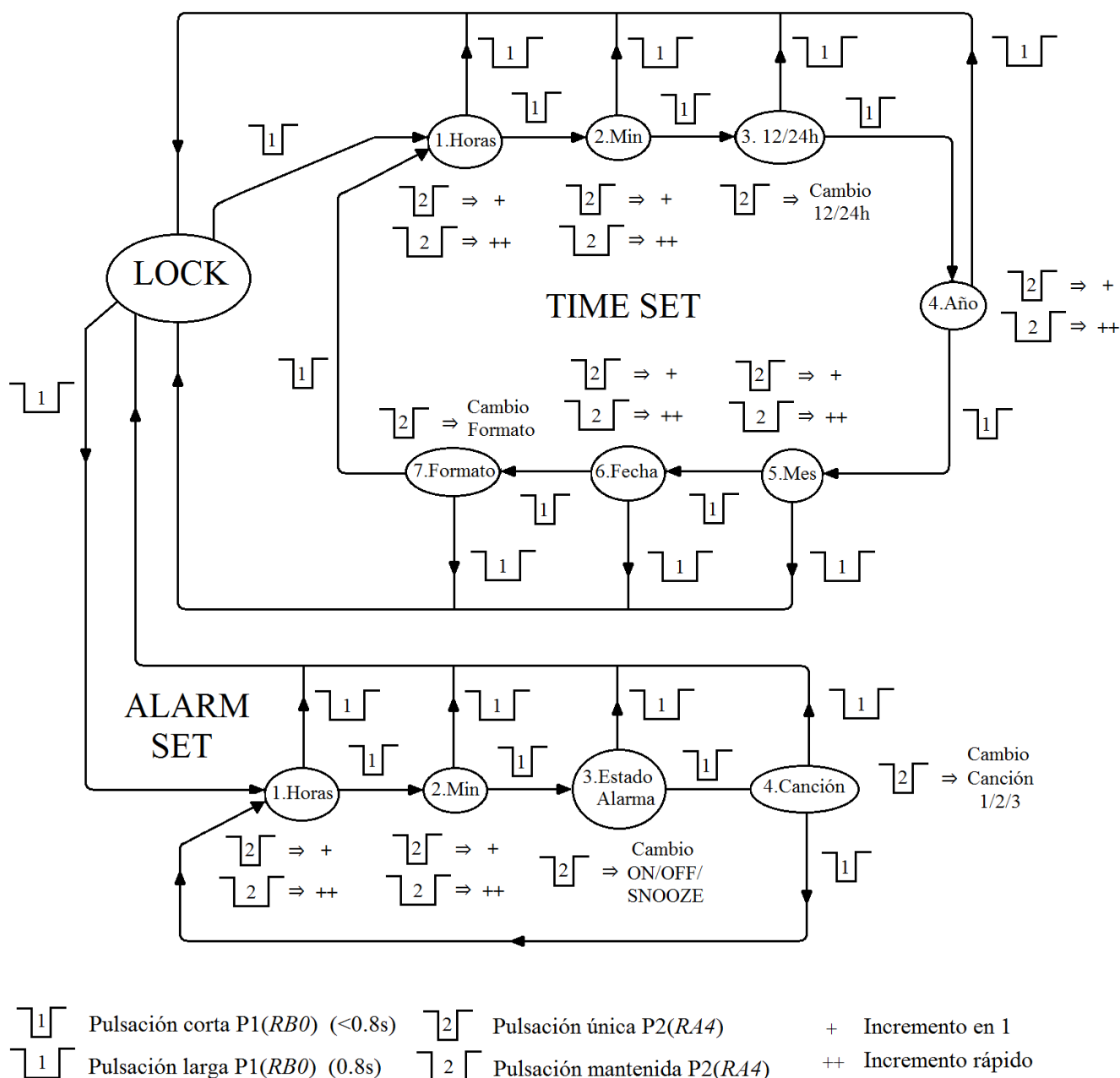


Figura 17. Diagrama configuración reloj (TIME SET) y alarma (ALARM SET).

3.5.3 Cómo apagar la alarma

Cuando la alarma se encuentra sonando, puede silenciarse pulsando cualquier botón. No obstante, hay diferencias en función del modo de la alarma seleccionado.

- **Alarma modo ON:** pulsar cualquier botón (no importa cuánto tiempo) apaga la alarma. Un pitido inmediatamente después confirma que la alarma ha sido apagada. La alarma no sonará hasta el día siguiente a la misma hora.

Si no se pulsa ningún botón, la canción escogida sonará por completo 3 veces y después el pitido confirmará el apagado. El objetivo de esta restricción es evitar molestar a los vecinos si el usuario se despierta antes y no se acuerda de desconectar la alarma.

- **Alarma modo SNOOZE:** la alarma sonará a la hora programada y 5 y 10 minutos más tarde, salvo que el usuario la apague. En cada ocasión la canción sonará a lo sumo 3 veces completas.

Cuando la alarma está sonando, pulsar cualquier botón silenciará la alarma, pero no la apagará (salvo que sea la tercera vez que la alarma está sonando). Para apagar por completo la alarma P1 o P2 deben mantener pulsados 5 s.

Se considera que el usuario debe estar completamente despierto para mantener pulsado un botón durante 5 s. Es indiferente si el usuario empieza a pulsar el botón cuando la alarma está sonando o en los 5 minutos que transcurren antes de que la alarma vuelva a sonar.

Cuando la alarma se apaga por completo (porque era la tercera vez o porque se ha mantenido un botón presionado durante 5 s) un pitido indica que la alarma ha sido, efectivamente, apagada. La alarma no sonará hasta el siguiente día a la misma hora.

La explicación sobre el apagado de la alarma se recoge de forma gráfica en el siguiente diagrama (ver leyenda en la parte inferior del mismo):

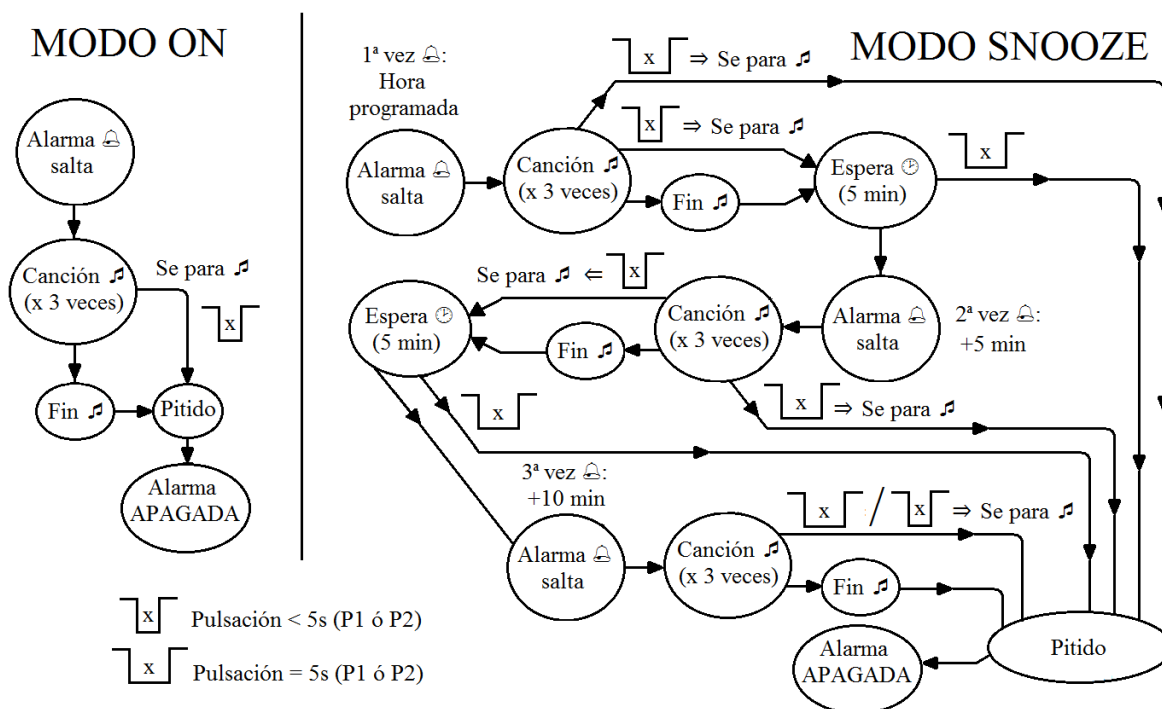


Figura 18. Diagrama apagado de la alarma: modo ON (izq.) y modo SNOOZE (der.).

3.6 Cálculo de del día de la semana

Aunque se han explicado una gran cantidad de aspectos de la programación del reloj despertador, todavía no se ha mencionado cómo se calcula el día de la semana.

Existen varios algoritmos y métodos para calcular el día de la semana conocida la fecha (día, mes y año). El algoritmo utilizado, conocido como método de las tablas, ha sido elegido por ser fácil de entender y programar. Se ha implementado por medio de la función *diasemana()*.

En primer lugar, a partir de la fecha, se calculan los siguientes coeficientes:

- *Coeficiente del día (a)*: valor del día (del mes).
- *Coeficiente del mes (b)*: se debe consultar la tabla inferior, que asocia a cada mes un coeficiente.

Enero	Febrero	Marzo	Abril	Mayo	Junio
6	2	2	5	0	3
Julio	Agosto	Sept.	Oct.	Nov.	Dic.
5	1	4	6	2	4

Tabla 15. Coeficientes para cada mes del año.

- *Coeficiente del año bisiesto (c)*: si el año es bisiesto y el mes es enero o febrero el coeficiente $c = -1$. En caso contrario $c = 0$.
- *Coeficiente del año (d)*: se toman las dos últimas cifras del año. d se obtiene sumando a ese valor la parte entera de su división entre 4.
- *Coeficiente del siglo (e)*: se toman las dos primeras cifras del valor del año y se multiplica el resto de su división entre 4 por 5. Para el siglo actual (siglo XXI, años 2000-2099), $e = 0$.

$$AÑO = Y_4Y_3Y_2Y_1$$

$$d = Y_2Y_1 + [Y_2Y_1 / 4]$$

$$e = (Y_4Y_3 \% 4) \cdot 5$$

con:

$f(x) = [x]$, función parte entera

$g(x, y) = x \% y$, función módulo

(resto de dividir x entre y)

En el programa sólo se permiten años del 00 al 99 sin especificar el siglo. Se ha considerado que el siglo siempre va a ser el XXI por lo que se ha omitido el cálculo del coeficiente e . Por lo tanto, el calendario sólo permite trabajar en el intervalo de años 2000-2099, que se ha estimado más que suficiente para el reloj despertador desarrollado.

Finalmente, se suman todos los coeficientes y se calcula el resto de dividir el resultado entre 7. La correspondencia entre el resto obtenido y el día de la semana es:

L	M	X	J	V	S	D
1	2	3	4	5	6	0

Tabla 16. Relación resto-día semana

3.7 Programa fuente

```
#include<htc.h>           //Se incluye la librería del microcontrolador a usar.
#include "lcd.h"           //Se incluye la librería de la pantalla LCD.
#include "i2c.h"           //Se incluye la librería para comunicación I2C.

#define LCD_RS RD4        //Se define LCD RS igual a RD4.
#define XTAL_FREQ 4000000 //Oscilador interno de 4MHZ.
_CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);
//Comandos de configuración.

unsigned char modo=0;      //Modo=0,LOCK; Modo=1,TIME SET; Modo=2,ALARM SET
unsigned char desp=0;      //variable despertador, desp=0 despertador en silencio;
                           //desp=1, desp. sonando; desp=2, "stand-by"(SNOOZE)
unsigned char i=1;        //contador asociado al selector
unsigned int j=0;          //contador asociado a TMR1 para temporiza 1s
unsigned char k;           //contador auxiliar utilizado en varios bucle for
unsigned char w=0;        //contador para limitar el n° de veces que se usa SNOOZE
unsigned short long p=0;   //contador para distinguir pulsación corta/larga
unsigned char fecha=1,fmax; //variables para la fecha y su valor máximo
const unsigned char ndmes[12]={31, //array con el número de días de cada mes
28,31,30,31,30,31,31,30,31,30,31}; //((año no bisiesto, mes de febrero 28 días)
unsigned char mes=1,annus=0; //variables para el mes y el año
unsigned char fr=0;        //variable para indicar el formato
                           //fr=0,DD/MM/YYT°C(europeo);fr=1,MM/DD/YY T°F(americano)
unsigned char alarma=0;    //estado alarma:0,OFF; 1,ON; 2,SNOOZE
unsigned char min A2;      //minutos de la alarma A2
unsigned char nc=1;        //n° canción seleccionada (1-3)
unsigned char t=0;         //contador utilizado para las notas de cada canción
const unsigned char tmax[3]={64,61,44}; //número de notas de cada canción

//Arrays de valores para crear los caracteres deseados
const unsigned char celsius[8]={0x16,0x09,0x08,0x08,0x08,0x09,0x06,0x00};
const unsigned char fahrenheit[8]={0x17,0x08,0x08,0x0E,0x08,0x08,0x08,0x00};
const unsigned char TIME[8]={0x1F,0x04,0x04,0x04,0x04,0x00,0x00,0x00};
const unsigned char ALARM[8]={0x06,0x09,0x0F,0x09,0x09,0x00,0x00,0x00};
const unsigned char SET[8]={0x0C,0x10,0x08,0x04,0x18,0x00,0x00,0x00};
const unsigned char campana[8]={0x04,0x0E,0x0A,0x0A,0x11,0x1F,0x04,0x00};
const unsigned char snooze[8]={0x1C,0x04,0x08,0x13,0x1D,0x02,0x03,0x00};
const unsigned char otilde[8]={0x02,0x04,0x0E,0x11,0x11,0x11,0x0E,0x00};

struct partitura{unsigned char notas[64];}; //Creación de la estructura tipo partitura,
//formada por un array (notas) de 64 elementos.
const struct partitura canc[6]={ //Creación de canc, array de 6 estructuras
//constantes tipo partitura, cada una con 64 elementos.

//canc se puede tratar como una matriz 6x64(6 filas x 64 columnas).
{6,2,2,2,2,0,2,2,0,6,6,6,6,6,0,2,6,2,2,2,2,0,2,2,0,6,6,6,6,6,2,11,
0,6,6,6,6,6,0,2,0,6,6,6,6,6,0,2,6,2,2,2,2,0,2,2,0,6,6,6,6,6,2,11},
{4,4,2,2,2,2,4,4,4,4,2,2,2,2,4,4,4,4,2,2,2,2,4,4,4,4,2,2,2,2,4,4,
4,4,2,2,2,2,4,4,4,4,2,2,2,2,4,4,4,4,2,2,2,2,4,4,4,4,2,2,2,2,4,4},
{10,10,7,7,8,7,6,8,7,6,5,7,6,5,4,4,4,5,6,7,8,6,7,8,7,6,5,4,6,7,5,6,
8,8,7,6,5,7,5,6,8,8,7,6,5,11,7,7,4,4,4,4,5,7,8,7,6,5,4,6,7},
{2,2,6,2,2,2,6,2,2,2,6,2,2,2,8,2,2,6,2,2,2,4,2,2,2,2,4,4,4,8,4,4,2,
2,2,2,4,4,4,4,2,2,2,2,4,4,2,2,6,2,2,2,4,2,2,2,2,4,4,4,8},
{6,6,3,6,6,3,6,3,6,9,9,6,6,3,6,6,9,9,9,9,6,9,9,9,9,6,6,9,9,9,3,6,
9,9,9,9,6,6,9,9,9,6},
{4,2,2,4,4,2,2,2,2,4,4,4,2,2,4,4,2,2,2,2,8,2,2,2,2,4,4,2,2,2,2,4,4,
2,2,2,2,4,4,2,2,2,2,8}
};

//DECLARACIÓN DE FUNCIONES
unsigned char leer(unsigned char rx);
void escribir(unsigned char ry, unsigned char dy);
unsigned char act(unsigned char reg,unsigned char v0,
unsigned char vmax, unsigned char x);
void act_hora(unsigned char al, unsigned char y);
void diasemana(void);
void act_Ta(void);
```

```

void charCGRAM(unsigned char n,
    const unsigned char values[]);
void cursor(void);
void nota(unsigned char u,unsigned char v);
void desp off();

//INTERRUPCIONES
static void interrupt isr(void){

    if(T0IF){
        //INTERRUPCIÓN RA4.
        if((desp==0)&&(modo==0)){
            // DESPERTADOR OFF y MODO 0 (LOCK).
            T0IF=0; TMR0=255; } //Se baja T0IF y se reconfigura TMR0.

        else if((desp==0)&&(modo==1)){
            // DESPERTADOR OFF y MODO 1 (TIME SET).
            while(T0IF||(~RA4)){ //Bucle 1ª pulsación RA4 o pulsación continua.
                delay ms(200); //Retardo de 200ms (rebotes).
                if(i==1){act(1,0,59,1);} //i=1, cambio de los minutos(+1).
                else if(i==2){act hora(0,1);} //i=2, cambio de la hora(+1).
                else if(i==3){for(k=0;k<3;k++){ //i=3, cambio de formato 12/24h
                    act hora(k,2);}
                else if((4<=i)&&(i<=6)){ //i=4 ó i=5 ó i=6.
                    fecha=act(4,'-','-',0); //Actualización fecha.
                    mes=act(5,'-','-',0); //Actualización mes.
                    annus=act(6,'-','-',0); //Actualización año.
                    if(i==4){annus=act(6,0,99,1);} //i=4, cambio de año(+1).
                    else if(i==5){mes=act(5,1,12,1);} //i=5, cambio de mes(+1).
                    if((mes==2)&&(annus%4==0)){fmax=29;}
                    else{fmax=ndmes[mes-1]};
                    if(fecha>fmax){fecha=1;escribir(4,1);}
                    if(i==6){fecha=act(4,1,fmax,1);} //i=6, cambio fecha(+1).
                    diasemana(); //Actualización día semana.
                    else if(i==7){if(fr==0){fr=1;} //i=7, modificación formato
                        else{fr=0;}}
            T0IF=0;TMR0=255; } } //Se baja T0IF y se reconfigura TMR0.

        else if((desp==0)&&(modo==2)){
            // DESPERTADOR OFF y MODO 2 (ALARM SET).
            while(T0IF||(~RA4)){ //Bucle 1ª pulsación RA4 o p. continua.
                delay ms(200); //Retardo de 200ms (rebotes).
                if(i==1){act(8,0,59,1);} //i=1, cambio minutos alarma(+1).
                else if(i==2){act hora(1,1);} //i=2, cambio hora alarma(+1).
                else if(i==3){ //i=3, cambio estado alarma.
                    if(alarma<2){alarma++;}else{alarma=0;} // (0:OFF, 1:ON, 2:SNOOZE)
                    if(alarma==0){CCP2CON=0x00; //Módulo CCP2 apagado.
                        escribir(0x0E,0x04);} //Alarma OFF: A1 y A2 OFF.
                    else{CCP2CON=0x04; //CCP2: captura, 1 flanco bajada.
                        escribir(0x0E,0x05);} //Alarma ON/SNOOZE: A1E=1
                    else if(i==4){ //i=4, cambio n° canción.
                        if(nc<3){nc++;}else{nc=1;} //n° de canciones: 3.
                        lcd goto(0x4B);lcd putch(0x30+nc); //Escritura n° de canción.
                        lcd goto(0x4B); } //Cursor en 0x4B.
            T0IF=0; TMR0=255; } } //Se baja T0IF y se reconfigura TMR0.

        else if(desp==1){
            // DESPERTADOR ON.
            nota(11,4); //Silencio de negra para detener la canción.
            if(alarma==1|| (alarma==2&&w==2)){ //Alarma ON o 2ª vez alarma SNOOZE:
                desp off(); } //Apagado del despertador.
            else if(alarma==2&&w<2){p=0; //1ª vez alarma SNOOZE: se inicializa p a 0.
                while((~RA4)&&(p<185000)){p++;} //Bucle RA4 pulsado y p<185000 (máx. Td 5s).
                if(RA4){desp=2;} //Pulsación corta: desp. pasa a "stand-by".
                else{desp_off();} } //Apagado del despertador.
            while(~RA4); //Se espera fin pulsación RA4.
            T0IF=0; TMR0=255; //Se baja T0IF y se reconfigura TMR0.

        else if(desp==2){
            // DESPERTADOR EN "STAND-BY".
            p=0; //Se inicializa p a 0.
            while((~RA4)&&(p<185000)){p++;} //Bucle RA4 pulsado y p<185000 (máx. Td 5s).
            if(~RA4){desp off(); //Pulsación larga: apagado del despertador.
                while(~RA4); //Se espera fin pulsación RA4,
            T0IF=0; TMR0=255; } //Se baja T0IF y se reconfigura TMR0.
    }
}

```

```

    }

    //INTERRUPCIÓN TMR1.
    if(TMR1IF){
        //j debe llegar a 2 para Td de aprox 1s.
        TMR1IF=0;TMR1H=11;TMR1L=220;
        //Se baja TMR1IF y se reconfigura TMR1.
        if(j<2){j++;}
        //Si j<2, se incrementa j.
        if(j==2){j=0;
        //Si j=2, se reinicia j a 0.
            lcd goto(0x0D);
            //Se dibujan los símbolos de la campana
            if(alarma==1){lcd putch(5);
            //y snooze en las posiciones 0x0D y 0x0E
            if(alarma==2){lcd putch(6);} }
            //en función del valor de alarma.
            else{lcd puts(" ");}

        if(modo<=1){
            // MODO 0 Ó MODO 1, ACTUALIZACIÓN RELOJ
            act(1,'-','- ',0); act hora(0,0);
            //Actualización minutos y horas.
            fecha=act(4,'-','- ',0);
            //Actualización fecha.
            mes=act(5,'-','- ',0);
            //Actualización mes.
            annus=act(6,'-','- ',0);
            //Actualización año.
            diasemana(); act Ta();
            //Actualización día de la semana y Tª.
            lcd goto(0x00);
            //Escritura en la posición 0x00.
            if(modo==1){
                lcd putch(2);lcd putch(4);} // "TS" para el Modo 1.
                else{lcd puts(" ");} // " " para el Modo 0.

            else{
                // MODO 2, ACTUALIZACIÓN ALARMA
                act(8,'-','- ',0); act hora(1,0);
                //Actualización minutos y hora alarma.
                lcd goto(0x00);
                //Escritura de "AS" en la posición 0x00.
                lcd putch(3); lcd putch(4);
                lcd goto(0x40);lcd puts(" Canci");
                //Escritura de "Canción" a partir de
                lcd putch(0x07);lcd puts("n:");
                //la posición 0x43.
                lcd putch(0x30+nc);
                //Actualización nº canción seleccionada.
                lcd puts(" ");
                }

            cursor();
        } //Cursor en la posición deseada.
    }

    if(CCP2IF){
        //INTERRUPCIÓN MÓDULO CCP2
        if(modo>=1){modo=0; i=1;}
        //Si Modo=1 ó 2, se vuelve a Modo 0, i a 1.
        if(alarma==2){
            //Alarma SNOOZE, se prepara A2.
            if(Leer(0x0F)%2){
                //AlF=1, supone 1ª vez que suena la alarma.
                escribir(0x0B,Leer(0x08));
                //Se copian min de A1 a A2.
                escribir(0x0C,Leer(0x09)); w=0;}
                //Se copian horas de A1 a A2, w a 0.
                else{w++;}
                //AlF=0, se suma 1 a w.
            if(w<2){
                //Si w<2, se programa alarma a los 5 min.
                min A2=act(0x0B,'-','- ',0);
                //Se lee el valor de los minutos de A2.
                if(min A2>=55){act hora(2,1);}
                //Si min A2>=55, sumar 5 min supone hora+1.
                for(k=0;k<5;k++){act(0x0B,0,59,1);}
                //Se suman 5 min a la alarma A2.
                escribir(0x0E,0x07);}
                //Se activa la alarma A2 (A2E=1).
            escribir(0x0F,0x00);
            //Se baja el flag AlF ó A2F.
            CCP2IF=0; desp=1;
            } //Se baja CCP2IF y se activa el despertador.
        }

    //PROGRAMA PRINCIPAL. Configuración puertos, módulos y periféricos.
    void main(void){
        i2c init(); lcd init();
        //Configuración módulo I2C y pantalla LCD.
        lcd_write(0x0C);
        //Cursor pantalla LCD invisible.

        ADCON1=0x06; TRISA=0xFF;
        //Config. PORTA como entradas digitales.
        TRISB=0xFF; TRISC=0xFB;
        //Config. PORTB y PORTC como entradas, salvo RC2.

        GIE=1; PEIE=1; T0IE=1;
        //Habilitación int. globales, periféricos y TMR0.
        OPTION_REG=0xBF;
        //INTEDG=0, interrupción RB0 por flanco de bajada.
        //TMR0 contador (flanco de bajada) sin predivisor.
        TMR0=255;
        //Config. TMR0 para que se desborde con 1 pulso.
        TMR1H=11; TMR1L=220;
        //Configuración de TMR1.
        T1CON=0x30;
        //TMR1 OFF, Td = [65536-(11*256+220)]*8*1us = 0.5s
        TMR1IF=0; TMR1IE=1; TMR1ON=1; //Flag TMR1 bajado, int.TMR1 habilitada, TMR1 ON.
    }

```

```

INTF=0; //Flag RB0 bajado(interrupción RB0 desactivada).
CCP2CON=0x04; //Módulo CCP2: modo captura, 1 flanco de bajada.
CCP2IF=0; CCP2IE=1; //Flag CCP2IF bajado, int. CCP2 desactivada.

lcd goto(0x06); lcd puts(":"); //Escritura ":" en 0x06 del LCD.
charCGRAM(0,celsius); charCGRAM(1,fahrenheit); //Caracteres celsius y fahrenheit.
charCGRAM(2,TIME); charCGRAM(3,ALARM); //Caracteres TIME (2) y ALARM (3).
charCGRAM(4,SET); charCGRAM(5,campana); //Caracteres SET (4) y campana (5).
charCGRAM(6,snooze); charCGRAM(7,otilde); //Caracteres snooze y otilde.

escribir(0x0E,0x04); //Int.DS3231 habilitada, A1 y A2 desactivadas.
escribir(0x0F,0x00); //Flags alarmas A1 y A2 del DS3231 bajados.
escribir(0x07,0x00); //AlM1=0, segundos de la alarma 1 a 0.
escribir(0x08,0x00); //AlM2=0, minutos de la alarma 1 a 0.
escribir(0x09,0x00); //AlM3=0, horas de la alarma 1 a 0.
escribir(0x0A,0x80); //AlM4=1.
escribir(0x0D,0x80); //A2M4=1.

//PROGRAMA PRINCIPAL. Bucle infinito
while(1){

    if(INTF){ //PULSACIÓN RB0 (INTF=1).
        if(desp==0){ // DESPERTADOR OFF
            p=0; //Se inicializa p a 0.
            while((~RB0)&&(p<27000)){p++;} //Bucle RB0 pulsado y p<27000(máx. Td 0.8s).
            if(modos==0){if(RB0){modos=1;} //Modo=0, pulsación corta; se pasa a Modo 1.
                else{modos=2;} } //Modo=0, pulsación larga; se pasa a Modo 2.
            else{ //Modo=1 ó Modo=2.
                if(RB0){ //Pulsación corta, se incrementa selector.
                    if(modos==1){if(i<7){i++;}else{i=1;}} //Modo 1, selector toma valores entre 1 y 7.
                    else{if(i<4){i++;}else{i=1;}}; } //Modo 2, selector toma valores entre 1 y 4.
                    else{modos=0; i=1;}; } //Pulsación larga, se vuelve a Modo 0.
                }
            }
            else if(desp==1){ // DESPERTADOR ON.
                nota(11,4); //Silencio de negra para detener la canción.
                if(alarma==1||(alarma==2&&w==2)){ //Alarma ON o 2ª vez alarma SNOOZE:
                    desp_off(); //Apagado del despertador.
                }
                else if(alarma==2&&w<2){p=0; //1ª vez alarma SNOOZE: se inicializa p a 0.
                    while((~RB0)&&(p<185000)){p++;} //Bucle RB0 pulsado y p<185000 (máx. Td 5s).
                    if(RB0){desp=2;} //Pulsación corta: despertador a "stand-by".
                    else{desp_off();} } //Apagado del despertador.
                }

            else if(desp==2){ // DESPERTADOR EN "STAND-BY".
                p=0; //Se inicializa p a 0.
                while((~RB0)&&(p<185000)){p++;} //Bucle RB0 pulsado y p<185000 (máx. Td 5s).
                if(~RB0){desp_off();} //Apagado del despertador.
            }

            while(~RB0); INTF=0; } //Espera fin pulsación (si RB0 pulsado).
            //Se baja el flag INTF.

        if(desp==1){ //DESPERTADOR ON, CANCIÓN.
            t=0; //Se inicializa t a 0.
            while((desp==1)&&(~INTF)&&
                (t<3*tmax[nc-1])){ //Bucle mientras desp. activado, flag bajado
                //y canción tocada menos 3 veces completas.
                nota(canc[2*(nc-1)].notas[t*tmax[nc-1]]); //Se toca nota t*tmax[nc-1] de canción nc,
                canc[2*(nc-1)+1].notas[t*tmax[nc-1]]); //empleando datos de la matriz can.
                t++; //Se incrementa en 1 el valor de t.
            }
            if(t==3*tmax[nc-1]){ //Si se ha tocado 3 veces la canción:
                if(alarma==1||(alarma==2&&w==2)){ //Alarma ON o 2ª vez alarma SNOOZE:
                    nota(11,4); //Silencio de negra para detener la canción.
                    desp_off(); //Apagado del despertador
                }
                else if(alarma==2&&w<2){desp=2;} } //Alarma SNOOZE: desp- pasa a "stand-by".
            }
        }
    } //Fin bucle programa principal.
} //Fin programa principal.

//CÓDIGO DE LAS FUNCIONES
unsigned char leer(unsigned char rx){ //Lectura registro D3231 (comunicación I2C)

```

```

    unsigned char dx; //Variable interna para los datos leídos.
    i2c start(); //Envío de la secuencia de start.
    i2c write(0xD0); //Dir. del esclavo(0b1101000)+ write bit(0).
    i2c write(rx); //Dirección rx del registro a leer.
    i2c restart(); //Envío de la secuencia de restart.
    i2c write(0xD1); //Dir. del esclavo(0b1101000)+ read bit(1).
    dx=i2c read(); //Lectura reg. rx y almacenamiento en dx.
    i2c sendnack(); //Envío secuencia de not acknowledgement.
    i2c stop(); //Envío de la secuencia de stop.
    return dx; //Devolución del valor dx.
}

void escribir(unsigned char ry, unsigned char dy){
//Escritura registro D3231 (comunicación I2C)
    i2c start(); //Envío de la secuencia de start.
    i2c write(0xD0); //Dir. del esclavo(0b1101000)+ write bit(0).
    i2c write(ry); //Dirección ry del registro a escribir.
    i2c write(dy); //Byte dy de datos de escritura.
    i2c stop(); //Envío de la secuencia de stop.
}

unsigned char act(unsigned char reg, //Función para actualizar min,
unsigned char v0, unsigned char vmax,unsigned char x){ //fecha, mes, año y min alarma.

    unsigned char p[9]={'-',0x08,'-', '-','-', //Array que asocia a cada reg. su respectiva
0x43,0x46,0x49,'-',0x08}; //posición en la pantalla LCD.
    unsigned char D, U, datos, valor; //Decenas(D), uds(U), contenido del registro
//y valor de la magnitud (min, s, fecha,...)
    datos=leer(reg); //Lectura del registro reg (escrito en BCD).
    if(reg==6){D=datos/16;} //Cálculo de D(sólo en el caso de los años,
else{D=(datos/16)%8;} //reg=0x06, la instrucción es distinta).
    U=datos%16; valor=10*D+U; //Cálculo de U y del valor de la magnitud.

    if(x==1){ //Entrada x=1, se incrementa valor en 1.
        if(valor<vmax){valor++;} //valor < valor máximo(vmax), se suma 1.
        else{valor=v0;} //valor = vmax, se reinicia a valor inicial(v0).
        D=valor/10; U=valor%10; //Se recalculan D y U.
        datos=16*D+U; //Se calcula el valor en BCD a escribir en reg.
        escribir(reg,datos); //Se actualiza el registro con el nuevo valor.
        if(reg==0x01){escribir(0x00,0x00);} //Si se actualizan min, se ponen s a 0.
    }

    if(fr){p[4]=0x48;p[5]=0x45;p[6]=0x4B;} //Para f.americano, cambian las posiciones
//del LCD asociadas a algunos registros.
    if(reg<=0x08){ //Sólo se muestran en LCD registros <= 0x08.
        lcd goto(p[reg]); lcd putch(0x30+U); //Escritura de las unidades en p[reg].
        lcd goto(p[reg]-1); lcd putch(0x30+D); //Escritura de las decenas en p[reg]-1.
        return valor; //Se devuelve el valor de la variable actualizada
    } //no se va a utilizar siempre).

void act_hora(unsigned char al, //Función para actualizar h o cambiar formato 12/24h.
unsigned char y){ //Empleada para la hora del reloj, AL1 y AL2.

    unsigned char Dh, Uh, h, datosh; //Dec. h, ud. h, valor h y valor del registro.
    unsigned char h12_24; //formato 24h, h12_24=0; formato 12h, h12_24=1
    unsigned char AMPM; //formato 12h: AM, AMPM=0; PM, AMPM=1
    const unsigned char hmax[2]={23,12}; //Valores mínimos h según formato 24/12h.
    const unsigned char hmin[2]={0,1}; //Valores máximos h según formato 24/12h.
    const unsigned char reg[3]={0x02,0x09,0x0C}; //Registros asociados a h

    datosh=leer(reg[al]); //Lectura del registro.
    h12_24=(datosh/64)%2; //Determinación del formato 12/24h.
    if(h12_24==0){Dh=(datosh/16)%4; Uh=datosh%16;} //F. 12h: cálculo Dh y Uh.
    else{Dh=(datosh/16)%2; Uh=datosh%16;} //F. 24h: cálculo de Dh, Uh y
        AMPM=(datosh/32)%2; // AMPM (hora AM, 0; hora PM, 1).

    if(y>0){h=10*Dh+Uh; //y=1 ó y=2, cálculo valor de la hora.
        if(y==1){ //y=1, se incrementa valor de la hora en 1.
            if(h<hmax[h12_24]){h++;} //h < hmax, se suma 1 a la hora.

```

```

        else{h=hmin[h12 24];} //h = hmáx, se reinicia a su valor mínimo.
    if((h12 24)&&(h==12)){if(AMPM==1){AMPM=0;} //Si formato es de 12h y h=12, se debe
        else{AMPM=1;}} //cambiar AM por PM o viceversa.
    else if(y==2){ //y=2, cambio formato 12/24h.
        if(h12 24==0){h12 24=1; //Caso 1: paso de formato 24h a 12h.
            if(h>12){h=h-12; AMPM=1;} //h=[13,23], se resta 12 y la hora es PM.
            else if(h==0){h=12; AMPM=0;} //h=0, h pasa a ser 12 AM.
            else if(h==12){h=12; AMPM=1;} //h=12, h pasa a ser 12 PM.
                else{AMPM=0;}} //h=[1,11], se deja igual y la hora es AM.
        else if(h12 24==1){h12 24=0; //Caso 2: paso de formato 12h a 24h.
            if((AMPM==0)&&(h==12)){h=0;} //Horas AM: igual excepto para 12AM(0:00).
            else if((AMPM==1)&&(h!=12)){h=h+12;}} //Horas PM: +12, salvo 12PM(12:00), igual.

    Dh=h/10; Uh=h%10; //Se calculan nuevos valores Dh y Uh.
    if(h12 24==0){datosh=16*Dh+Uh;} //Valor a escribir en registro: formato 24h
        else{datosh=64+32*AMPM+16*Dh+Uh;} //o formato 12h.
    if((al==0)&&(y==1)){escribir(0x00,0x00);} //Si es hora reloj, se ponen s a 0.
    escribir(reg[al],datosh); //Escritura del nuevo valor del registro.

    if((al==0)||((al==1)&&(y<=1))){ //Escritura en LCD (sólo en ciertos casos).
        lcd_goto(0x0A); //Posición 0x0A de la pantalla LCD.
        if(h12 24==0){lcd_puts(" ");} //Formato 24h: se escribe " ".
            else if(AMPM==0){lcd_puts("AM");} //Formato 12h: se escribe "AM" ó "PM".
                else {lcd_puts("PM");}
        lcd_goto(0x05);lcd_putch(0x30+Uh); //Escritura de Uh en 0x05.
        lcd_goto(0x04);lcd_putch(0x30+Dh); //Escritura de Dh en 0x04.
        if(y==2){lcd_goto(0x0A);} //Si se cambia el formato, retorno a 0x0A.
    }

void diasemana(void){
    unsigned char a, b, c, d, dia; //Variables internas.
    const unsigned char cfm[]={6,2,2,5,0,3,5,1,4,6,2,4}; //Coeficientes de cada mes.
    const unsigned char ldsem[7]={'D','L','M','X','J','V','S'}; //Letras día de la semana.
    const unsigned char ldw1[7]={'S','M','T','W','T','F','S'}; //3 letras día de la semana
    const unsigned char ldw2[7]={'u','o','u','e','h','r','a'}; //(en inglés).
    const unsigned char ldw3[7]={'n','n','e','d','u','i','t'};

    a=fecha; b=cfm[mes-1]; d=annus+annus/4; //a:coef.día, b:coef.mes, d:coef.año
    if((annus%4)==0)&&(mes<=2)){c=1;}else{c=0;} //c:coeficiente de año bisiesto

    dia=(a+b-c+d)%7; //Cálculo del resultado.
    escribir(0x03,dia); //Escritura del día en el registro 0x03.
    lcd_goto(0x40); //Escritura del día en la pantalla LCD.
    if(fr==0){lcd_putch(ldsem[dia]);lcd_puts(" ");} //fr=0, letra en castellano.
        else{lcd_putch(ldw1[dia]);lcd_putch(ldw2[dia]); //fr=1, 3 letras en inglés.
            lcd_putch(ldw3[dia]);lcd_puts(" ");}
    if(modos==1){cursor();} //Modo=1, se llama a la función cursor.
}

void act Ta(void){ //Función para calcular la Tª en °C o en °F.
    unsigned char datosT, T, DT, UT, dt; //Variables internas asociadas a Tª.
    lcd_goto(0x44+fr*2);lcd_puts("/"); //Esta función también actualiza la posición
    lcd_goto(0x47+fr*2);lcd_puts("/"); //de "/" de la fecha según el formato.
    datosT=leer(0x11); //Lectura del registro de temperatura.
    DT=datosT/10; UT=datosT%10; dt=leer(0x12)/64; //Cálculo de decenas(DT), unidades(UT) y
        //parte decimal(dT) de la temperatura.
    if(fr==0){ //fr=0, formato europeo.
        lcd_goto(0x4A);lcd_puts(" "); //Escritura en LCD de "DTUT.dT°C".
        lcd_putch(0x30+DT);lcd_putch(0x30+UT); //dT posee un valor entre 0 y 3.
        lcd_puts("."); //Por simplicidad la parte decimal de la Tª
    if(dt<2){lcd_puts("0");}else{lcd_puts("5");} //se considera .0 si dT<2 o .5 si dT>=2.
        lcd_putch(0x00); //Carácter "°C".
    else{ //fr=1, formato americano.
        T=(9*(10*DT+UT)/5+32); //Conversión Tª: °F = 9/5*(°C) + 32
        DT=(T/10); UT=(T-10*DT); //Cálculo DT y UT en °F.
        lcd_goto(0x4C);lcd_puts(" "); //Escritura en LCD de "DTUT°F".
        lcd_putch(0x30+DT);lcd_putch(0x30+UT);
        lcd_putch(0x01); //Carácter "°F".
    }
}

```



```

void charCGRAM(unsigned char n,
               const unsigned char values[]){          //Dibujo de un carácter en CGRAM.
    unsigned char z;                                //Variable interna.
    if(n<=7){LCD_RS=0;lcd write(0x40+8*n); //LCD RS=0, selección pos. CGRAM(0x40+8*n).
        for(z=0;z<8;z++){lcd putch(values[z]);} //Introducción del carácter en CGRAM.
    }
}

void cursor(void){
    unsigned char posml[]={0x08,0x05,0x0A,0x49,0x46,0x43,0x4F}; //Posiciones cursor Modo 1.
    const unsigned char posm2[]={0x08,0x05,0x0E,0x4B}; //Posiciones cursor Modo 2.
    if((fr==1)&&(3<i)&&(i<7)){ //Posiciones del cursor en Modo 1
        posml[3]=0x4B;posml[4]=0x45;posml[5]=0x48;} //dependientes del formato.
        if(modos==0){lcd goto(0x00);lcd write(0x0C);} //Modo 0: se desactiva el cursor.
    else if(modos==1){lcd goto(posml[i-1]); lcd write(0x0E);} //Modo 1: cursor activado.
    else if(modos==2){lcd goto(posm2[i-1]); lcd write(0x0E);} //Modo 2: cursor activado.
    }

//Clave notas:MiH=0,ReH=1,DoH=2,Si=3,Sib=4,La=5,Sol=6,Fa=7,MiL=8,ReL=9,DoL=10,Sil=11.
//Clave de tiempos: número de retrasos de 125 ms a los que equivale esa duración.
//bpunt(3tiempos)=12, 2.5tiempos=10, b=8, npunt=6, n=4, cpunt=3, c=2, s=1.

//Interpretación de una nota musical
void nota(unsigned char u,unsigned char v){          //u:código nota, v:duración nota.
    const unsigned char PN[]={94,106,119,          //PN = período notas musicales (T)
        126,133,141,158,178,188,212,238,250}; // (dividido entre 16 y expresado en us).
    const unsigned char TON[]={48,54,60, //TON = D*T para cada nota, con D = 0.5
        64,67,71,80,90,95,107,120,0}; // (dividido entre 16 y expresado en us).

    unsigned char w; //Contador auxiliar.
    CCP1CON=0x0C; CCP1L=TON[u]; //CCP1 modo PWM, M2=0 y CCP1L = TON.
    TMR2=0x00; PR2=PN[u]; //Se inicia TMR2 a 0 y PR2 = T(período).
    T2CON=0x06; //P = 16 (para compensar /16), TMR2ON=1.
    for(w=0;w<v;w++){__delay_ms(125);} //Retardo de 125 ms. Se ejecutará v veces.
    TMR2ON=0x00; CCP1CON=0x00; //Se desactiva TMR2 y se resetea CCP1.
    }

void desp off(){ //Función para apagar el despertador
    desp=0; //Despertador se apaga por completo.
    nota(7,2); //Pitido para confirmar apagado.
    escribir(0x0E,0x05); //Se deja activa A1(A1E=1) para la misma hora
    //del día siguiente. Alarma A2 desactivada (A2E=0).
    escribir(0x0F,0x00); //Se baja el flag A1F ó A2F según corresponda.
    }

```

3.8 PCB

Se incluyen en este apartado imágenes en 2D y 3D del PCB diseñado. Las referencias de los componentes se pueden consultar en la **Tabla 4**.

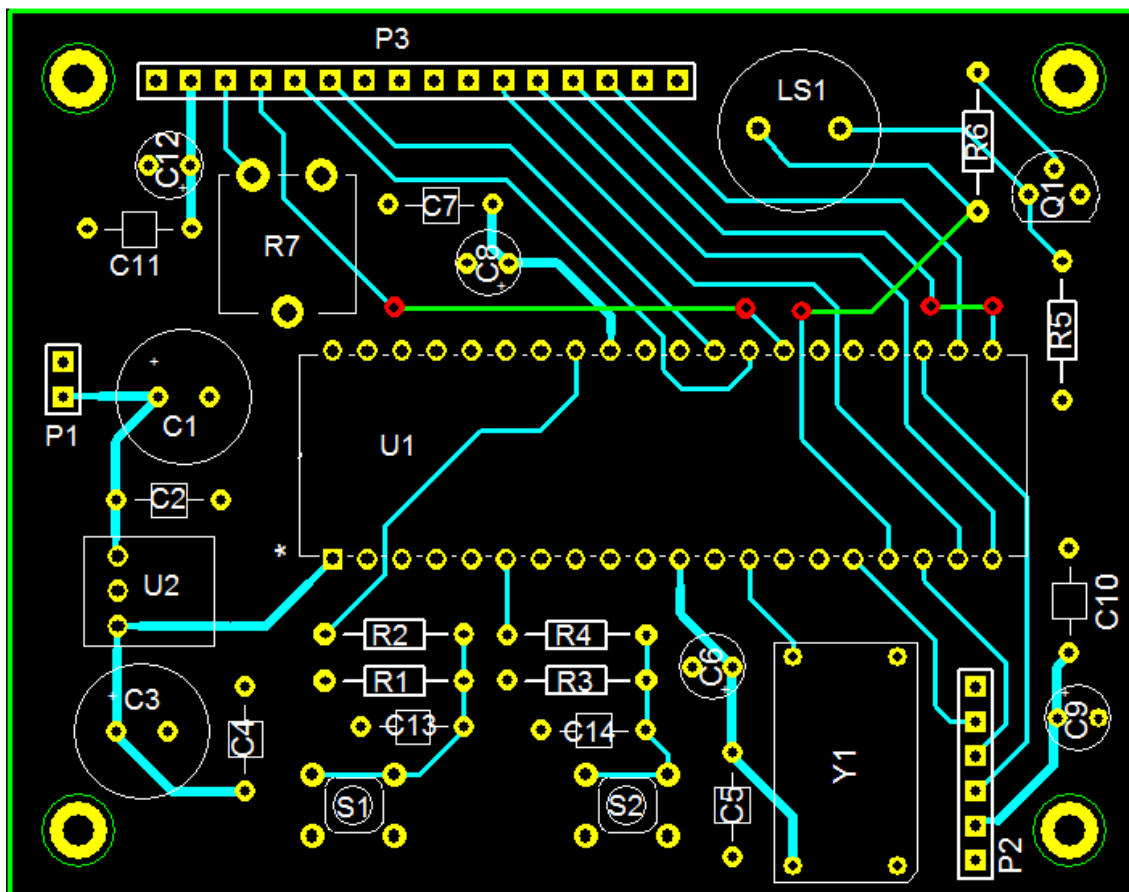


Figura 19. Vista del PCB sin planos a escala 2/1.

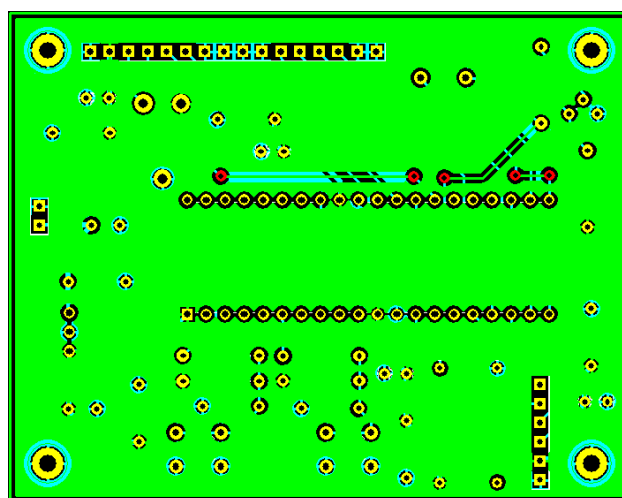


Figura 20. Vista del PCB con planos a escala 1/1.

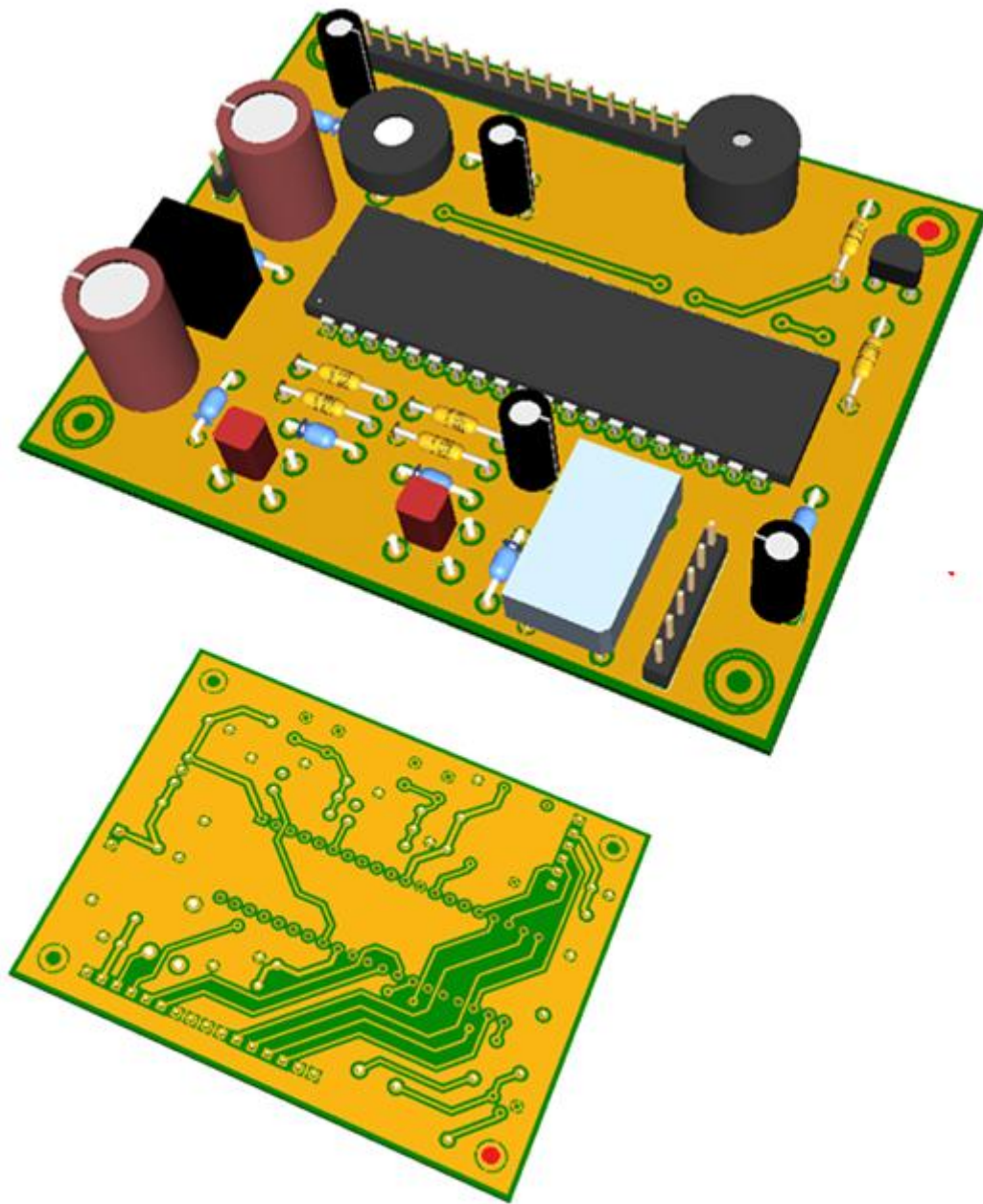


Figura 21. Vista 3D del PCB (cara superior e inferior)

El punto rojo permite identificar las esquinas correspondientes de ambas vistas del PCB. La vista de la cara inferior se obtiene dando la vuelta al PCB por el lado más largo más cercano al oscilador.

3.9 Diseño 3D

El diseño 3D posee 3 partes principales:

➤ Caja

La caja cuenta con los siguientes compartimentos:

- *Compartimento izquierdo*: puede contener la batería. Sin embargo, se puede utilizar el transformador de 9 V en su lugar, con su conector encajado en un agujero adyacente al compartimento de la pila. Otro agujero permite a los cables procedentes del conector llegar al PCB.
- *Compartimento central*: alberga el PCB. El PCB posee 4 agujeros (uno en cada esquina) para unir el PCB a las 4 pequeñas columnas que tiene el compartimento central. Deben usarse tornillos de métrica M2 o inferior. El compartimento central también tiene varios agujeros pequeños en la pared más cercana al zumbador para mejorar el sonido.
- *Compartimento derecho*: cuenta con una plataforma con 3 pequeños agujeros para anclar el circuito integrado DS3231, RTC empleado en el proyecto.

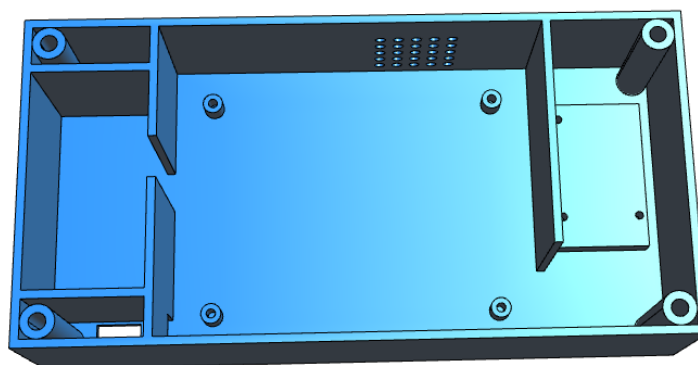


Figura 22. Diseño 3D de la caja.

➤ Tapa

La tapa dispone de un agujero rectangular donde encajar el display LCD y 2 agujeros circulares para los pulsadores. La tapa y la caja pueden unirse con tornillos de métrica M4.

En función de las dimensiones exactas del agujero rectangular, el display LCD encajará perfectamente o se caerá. En caso de que la segunda situación tenga lugar, la tapa cuenta con un saliente en su parte inferior donde se pueden introducir soportes para el display LCD (ver imagen situada a la derecha).

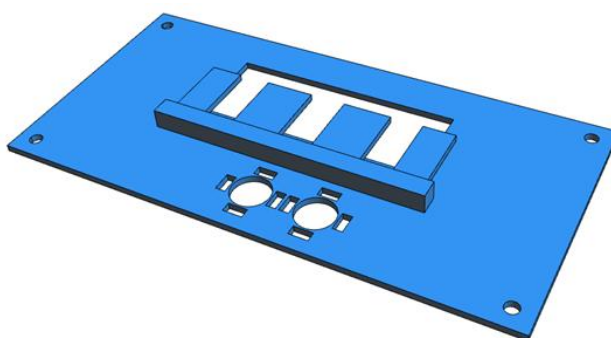


Figura 23. Montaje de la tapa (vista inferior).

➤ Pulsadores

Se han efectuado diseños específicos para ambos pulsadores, con los números 1 y 2 grabados en su superficie. Para ensamblar cada pulsador se requiere un muelle (de en torno a 1 cm de diámetro) y un tornillo (se recomienda la métrica M4).

Cuando no se presiona el pulsador (pieza 3D), la cabeza del tornillo no debe tocar el pulsador (componente through-hole) situado en el PCB. Cuando se presiona el pulsador, la rosca debe deslizarse a través del agujero en la caja del pulsador y la cabeza debe tocar el pulsador (componente through-hole) localizado en el PCB. Cuando la presión desaparezca el muelle debe ser capaz de hacer que el tornillo vuelva a la posición anterior.

Es probable tener que utilizar varios muelles y tornillos (prestando también atención a su longitud) antes de encontrar los más adecuados.

Tras ensamblar todas las piezas, el resultado final es:

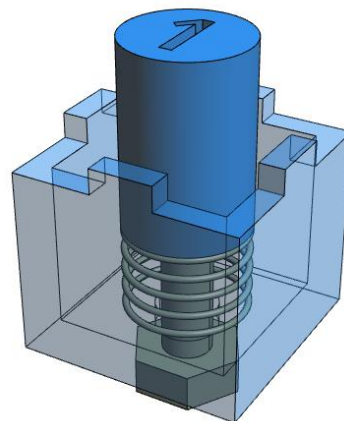


Figura 24. Montaje pulsador.

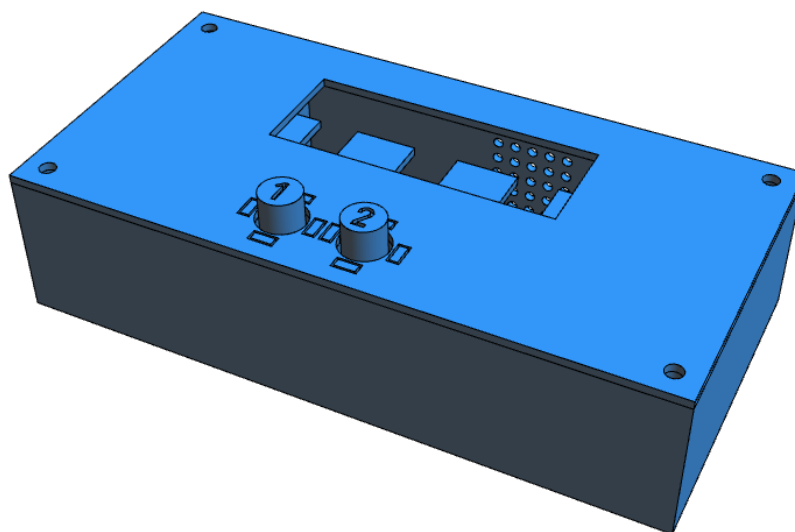


Figura 25. Montaje final Reloj Despertador.

3.10 Multimedia

Se ha publicado un artículo en inglés sobre este proyecto bajo el título **“WAKING YOURSELF UP – design your own alarm-clock from scratch”** en el sitio web DesignSpark, vinculado a la empresa RS Components.

En el artículo se incluyen todos los archivos vinculados al proyecto: código fuente, diseño del esquemático, diseño del PCB y diseño 3D.

➤ [Enlace al artículo](#)

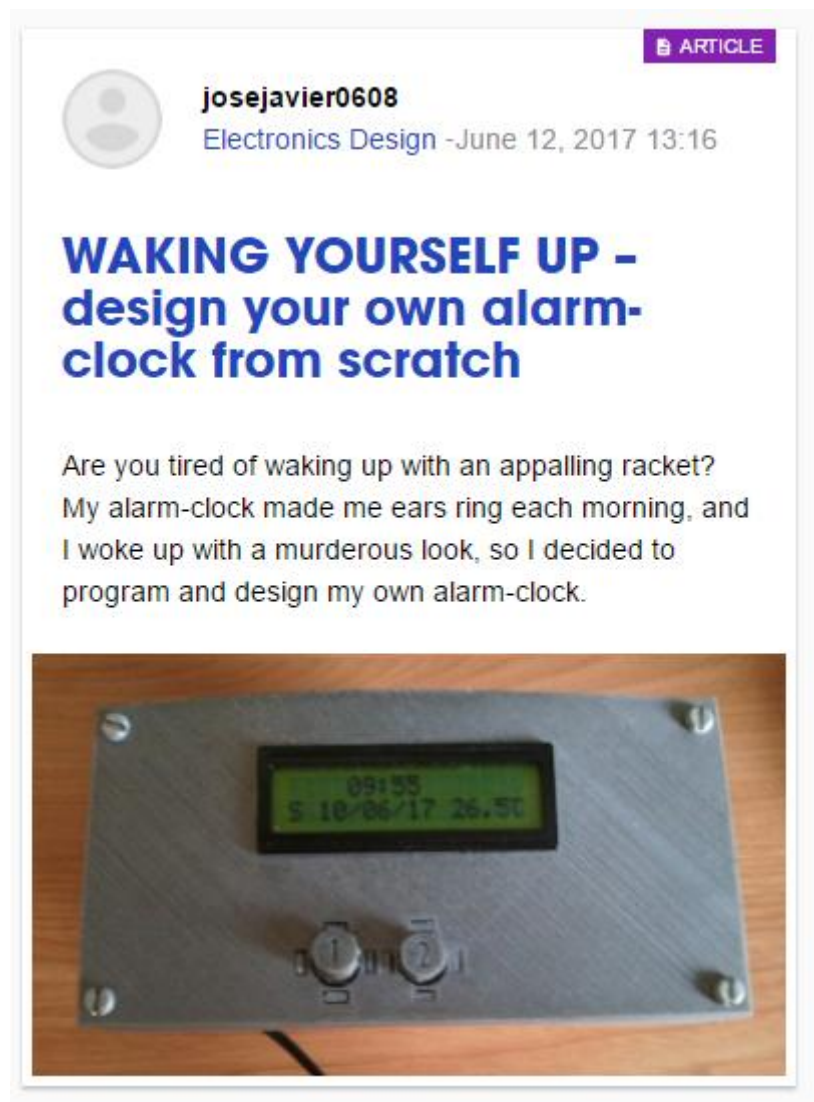


Figura 26. Cabecera de presentación del artículo.

4 Laser-shooting Robot

4.1 Introducción

Este proyecto está inspirado en las máquinas recreativas que cuentan con pistolas láser con las que acertar a objetivos que se van iluminando sucesivamente. No obstante, esta versión carecerá de dianas y se modificará el principio de funcionamiento.

El usuario dispondrá de un joystick para controlar el movimiento de una estructura de 2 grados de libertad accionada por servomotores, en cuya parte superior se situará el láser. Con el movimiento del joystick en el eje X se controlará la rotación de la estructura (1^{er} grado de libertad) y con el movimiento en el eje Y su inclinación (2^o grado de libertad). La pulsación del joystick encenderá el láser, que se apagará cuando se deje de pulsar.

4.2 Diseño esquemático

En este apartado se indican las conexiones requeridas por cada uno de los periféricos empleados en este proyecto. Para las conexiones del bloque de alimentación y el microcontrolador, consultar el apartado 2.1. No obstante, en esta ocasión también se van a comentar ciertos aspectos relativos al bloque de alimentación (ver ★ al final del apartado).

Los periféricos utilizados en este proyecto son:

- **2 servomotores SG90:** se hallan fijados en una estructura cuyo propósito original era sujetar y controlar una webcam. El primer servomotor permite la rotación (giro φ alrededor del eje Z, denominado alabeo, giro o roll en Robótica) de esta estructura y el segundo su inclinación (giro ψ alrededor del eje X, denominado guiñada, desviación o yaw en Robótica).

Cada servomotor posee 3 terminales: *PWM* (terminal de control, cable de color naranja), *V_{CC}* (5 V, rojo) y *GND* (0 V, negro). La señal PWM (Pulse Width Modulation, es decir, Modulación por Anchura de Pulsos) debe tener un periodo de 20 ms (frecuencia de 50 Hz) y un *T_{ON}* (ciclo de trabajo x período) comprendido entre 1 y 2 ms para que el servomotor posea un rango de giro de 180°.



v.20/06/17 **Figura 28.** Servomotor SG90 (apreciar el color del cable asociado a cada terminal).

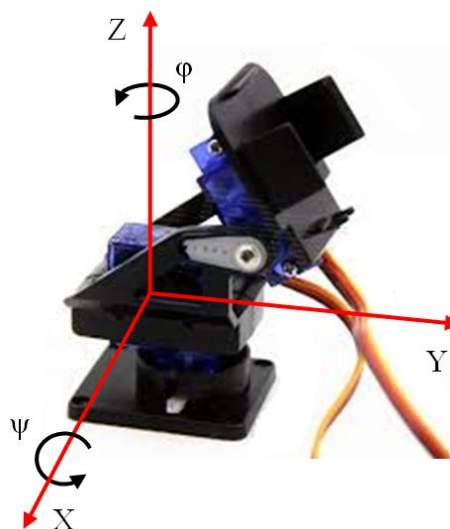


Figura 27. Estructura que integra los 2 servomotores, con sus giros indicados.

Los terminales *PWM* de los servomotores se conectan a *RC2* (servomotor encargado de la rotación) y a *RC1* (servomotor responsable de la inclinación) del microcontrolador, pines que se corresponden con las salidas de los módulos CCP1 y CCP2 respectivamente.

- **Joystick:** controla el giro de los servomotores y el encendido del láser. Dispone de 5 terminales: 2 pines de alimentación (+5V y GND), 2 pines analógicos (VRx y VRy) y un pin digital (SW).

La señal en los terminales analógicos está comprendida entre 0 y 5 V y es proporcional a la posición del joystick en los ejes X e Y respectivamente. Su funcionamiento se basa en la utilización de potenciómetros. El valor de estas señales se adquiere mediante el conversor analógico-digital (CAD) conectando los terminales a las entradas analógicas RA2 (VRx) y RA3 (VRy).

La señal en el terminal SW (switch) es un '0' lógico (0 V) si se presiona el pulsador y un '1' lógico en caso contrario. Se conecta a la entrada RA4 y requiere una resistencia de pull-up, ya que el joystick carece de ella.



Figura 29. Joystick.

- **Láser:** se sitúa en la parte superior de la estructura con los 2 servomotores y cuenta con 3 terminales: Sig, V_{CC} (5 V) y GND. Sig es el terminal de control. Si este pin recibe un '1' lógico el láser se enciende y si recibe un '0' se apaga. Se conecta al pin RC0 del microcontrolador.



Figura 30. Láser.

- ★ **Bloque de alimentación:** inicialmente se pretendía utilizar una batería de 9 V como fuente de alimentación. Sin embargo, la corriente demandada por los servomotores (en torno a 120 mA cada uno) cuando se mueven supone una importante caída de tensión en el sistema de alimentación.

Como resultado, se produce un brown-out reset y los servomotores vuelven a su posición inicial. Este problema tenía lugar cada vez que uno de los servomotores se movía y no era admisible.

Aunque se han probado diferentes soluciones (condensadores de reserva mayores para los servomotores, deshabilitar el brown-out reset en el microcontrolador) finalmente se ha decidido sustituir la fuente de alimentación, empleando el transformador de 9 V en lugar de la batería.

En el momento de finalizar la redacción de este trabajo, se considera que uno de los factores que también puede haber influido en este problema es la corriente máxima del regulador (0.5 A). Teóricamente es suficiente, pero ciertos indicios apuntan a esta causa. No obstante, no se ha comprobado experimentalmente.

Por último, también se añade al bloque de alimentación un interruptor de palanca, que interrumpe la línea de 9 V y permite al usuario encender y apagar todo el circuito de forma sencilla.



Figura 31. Interruptor de palanca.

4.2.1 Esquemático

Se muestra en esta página el esquemático del proyecto.

El transformador de 9 V y su conector, el interruptor de palanca, los servomotores y el láser no aparecen en el esquemático al no ser elementos integrados en el PCB. En cambio, sí que se indican los pines a los que se conectan (componentes soldados al PCB), identificados con el nombre del terminal correspondiente.

La anotación (N.C.) significa que el pin adyacente a la misma no se encuentra conectado. Concretamente, se trata del pin “1” del oscilador.

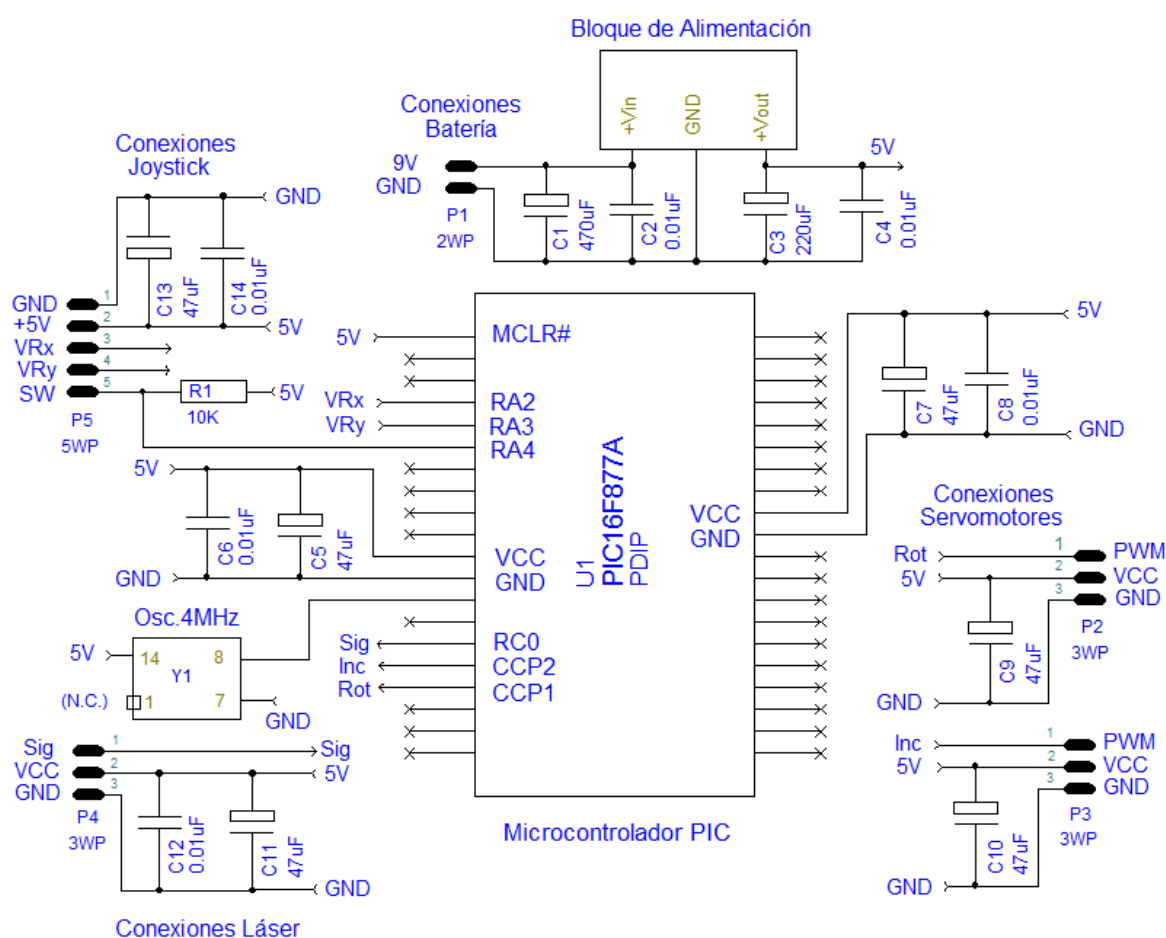


Figura 32. Esquemático (Laser-shooting Robot).

4.2.2 Listado de componentes

Se incluye debajo de estas líneas una lista con todos los componentes empleados en el proyecto. En el caso de los elementos que forman parte del PCB se indica su referencia, una breve descripción, el tipo de empaquetado y su cantidad. Los componentes que no disponen de referencia son aquellos que no se sitúan en el PCB (alimentación y periféricos).

Referencia	Descripción	Valor	Empaquetado	Uds
P1	Conector alimentación (2 terminales)		DSC	1
P2-4	Conector servomotor/láser (3 terminales)		DSC	3
P5	Conector joystick (5 terminales)		DSC	1
R1	Resistencia de pull-up	10 k Ω	DSC	1
C1	C _{IN} regulador	470 μ F	DSCV	1
C3	C _{OUT} regulador	220 μ F	DSCV	1
C2, C4, C6, C8, C12, C14	Condensador de desacoplo	0.01 μ F	DSC	6
C6, C8-9, C12	Condensador de reserva	47 μ F	DSCV	4
U1	Microcontrolador PIC16F877A		PDIP	1
U2	Regulador de conmutación R78E5.0-0.5		SIP3	1
Y1	Oscilador f = 4MHz		DIP	1
-	Interruptor de palanca SPDT			1
-	Transformador 9 V			1
-	Conector DC hembra			1
-	Soporte webcam			1
-	Servomotor SG90			2
-	Láser			1
-	Joystick			1

Tabla 17. Listado de componentes proyecto Laser-shooting Robot.

4.3 Servomotores

Se dedica un apartado de la explicación de este proyecto a los servomotores porque resultan los elementos más complejos de manejar. A pesar de que su control no es complicado, es necesario tener en cuenta una serie de detalles si se desea que este sea óptimo.

- *Funcionamiento real de los servomotores:* como es habitual, el funcionamiento real difiere del ideal. Si se utiliza una señal de control de 50 Hz con un T_{ON} entre 1 y 2 ms el “rango de disparo” es, en general, muy inferior a 180°.

Para lograr un rango de 180°, se debe usar un generador de funciones para “calibrar” el funcionamiento de los servomotores. No obstante, hay que evitar llevar al servomotor más allá de sus límites, porque se pueden dañar sus engranajes.

En este proyecto, el primer servomotor utiliza un T_{ON} comprendido entre 0.6 ms y 2.4 ms, que en el caso del segundo oscila entre 0.7 ms y 2.4 ms. Obviamente, las tolerancias de cada servomotor hacen que el intervalo válido para su ciclo de trabajo sea ligeramente distinto.

- *Posiciones límite:* una vez “calibrados” los servomotores, resulta conveniente restringir su movimiento por software. Por ejemplo, cuando el primer servomotor alcanza un T_{ON} de 0.6 ms, que se corresponde aproximadamente con la posición derecha extrema, todas las órdenes del usuario para girar a la derecha deben ser ignoradas. De este modo se puede garantizar que el servomotor no será forzado a girar más allá de sus límites.
- *Realimentación de la posición:* en realidad, no se emplea realimentación, pero se necesita asegurar que cada servomotor alcance la posición deseada. Esto significa que la señal PWM (con su correspondiente ciclo de trabajo) debe ser enviada continuamente al servomotor, incluso si no se producen cambios en su posición.

De esta forma, si el giro del servomotor es forzado externamente (sin usar el joystick, acción desaconsejada ya que puede dañar el servomotor) dado que la señal PWM se encuentra alimentando continuamente el servomotor, este retornará de forma inmediata a su posición previa.

- *Servomotor bloqueado:* finalmente, puede suceder que tras haber conseguido que el servomotor funcione correctamente, este deje de responder a la señal PWM enviada sin causa aparente.

La razón más probable es que un giro brusco (o haber forzado el giro del servomotor) haya ocasionado que dos engranajes se hayan bloqueado o que algún diente de los mismos se haya dañado.

La solución suele pasar por desmontar el servomotor, quitar todos los engranajes y alimentar el servomotor con el generador de funciones. Se van añadiendo sucesivamente los engranajes y se repite la comprobación, lo que permite detectar si existen problemas con alguno de ellos o si simplemente se trataba de un bloqueo.

Si algún engranaje se encuentra estropeado puede que aún y todo se pueda utilizar. En caso contrario habrá que sustituirlo o si se carece de recambio directamente reemplazar el servomotor.

4.4 Configuración del microcontrolador

En este apartado se recoge la configuración de los puertos y de los módulos del microcontrolador utilizados en el proyecto. Los módulos que se han empleado son: el temporizador TMR0, el conversor analógico-digital (CAD) y los 2 módulos CCP (CCP1 y CCP2), configurados en modo Comparación (se emplea el temporizador TMR1).

4.4.1 Configuración de los puertos

Los terminales del microcontrolador utilizados son:

ENTRADAS			SALIDAS	
Pin	A/D	Descripción	Pin	Descripción
RA2	A	Entrada analógica VRx	RC0	Salida digital ON/OFF Láser
RA3	A	Entrada analógica VRy	RC1 (CCP2)	Señal de control PWM Servomotor 2
RA4	D	Entrada digital SW	RC2 (CCP1)	Señal de control PWM Servomotor 1

Tabla 18. Entradas y salidas microcontrolador (Laser-shooting Robot).

Por tanto, los puertos se configuran del siguiente modo:

Registros

ADCON1 = 0x80; \Rightarrow PCFG = 0b0000 \Rightarrow

\Rightarrow RA0-RA3, RA5 y RE0-RE2; terminales analógicos,
V_{REF+} = 5 V y V_{REF-} = 0 V para conversión A/D.

TRISA = 0xFF; \Rightarrow PORTA: todos los terminales configurados como entradas.

TRISC = 0x00; \Rightarrow PORTC: todos los terminales configurados como salidas.

4.4.2 Configuración TMR0

TMR0 se encuentra gestionado por interrupción. Se configura para lanzar una conversión A/D cada 0.1 s. Para ello se temporizan 4 ms con TMR0 y se utiliza un contador auxiliar (*contTMR0*) de valor inicial 25,

El contador se decrementa en 1 cada vez que salta la interrupción (momento en el que hay que volver a reconfigurar TMR0). Cuando este contador llega a 0, se lanza la conversión A/D y se reinicia el contador a 25.

Registros

GIE = 1; TMR0IE = 1; \Rightarrow Interrupción global e interrupción TMR0 habilitadas.

OPTION_REG = 0x03; \Rightarrow T0CS = 0 \Rightarrow TMR0 configurado como temporizador
 \Rightarrow PSA = 0 \Rightarrow Predivisor asignado a TMR0.
 \Rightarrow PS2-0 = 011 \Rightarrow Predivisor TMR0 = 16

TMR0 = 0x06; (valor escrito cada vez que se reconfigura TMR0)

Temporización TMR0: $T_d = (256 - TMR0) * P * (4/F_{osc}) =$
 $= (256 - 6) * 16 * (4/4\text{MHz}) = 4 \text{ ms}$

4.4.3 Configuración conversor analógico-digital (CAD)

Cada 0.1 s se enciende el módulo CAD y se lanza una conversión analógica-digital, que se gestiona por interrupción. Tras finalizar la conversión, se apaga el módulo CAD, que no se vuelve a encender hasta que transcurran de nuevo 0.1 s.

El canal seleccionado para la conversión no es siempre el mismo, se alterna entre el canal 2 (RA2, lectura del valor de VRx) y el canal 3 (RA3, lectura del valor de VRy).

Registros

GIE = 1; PEIE = 1; ⇒ Int. global e interrupciones por periféricos habilitadas.
ADIE = 1; ⇒ Interrupción conversor A/D habilitada
ADCON1 = 0x80; ⇒ ADFM = 1 ⇒ Ajuste a la derecha (byte menos significativo en ADRESL, 2 bits más significativos en ADRESH).
 ⇒ ADSC2 = 0
ADCON0 = 0x50 / 0x58; ⇒ ADSC1-0 = 01 ⇒ TAD = FOSC/8 = 2 μs > 1.6 μs
 ⇒ CHS2-0 = 010 ó CHS2-0 = 011 ⇒
 ⇒ Canal seleccionado: 2 (RA2) ó 3 (RA3)

Pasos de la conversión A/D

1. Se enciende el conversor A/D. **ADON = 1**
2. Se espera el tiempo de adquisición T_{ACQ} (en torno a 20 μs). **Se esperan 25 μs.**
3. Se inicia la conversión A/D. **GO_DONE = 1**
4. Se espera a que salte la interrupción (ADIF = 1).
5. ADIF = 1 ⇒ Se baja el flag (**ADIF = 0**) y se apaga el conversor (**ADON = 0**).
6. Se lee el resultado de la conversión A/D. **256*ADRESH + ADRESL**
La resolución del conversor A/D es de 10 bits, luego el resultado de la conversión es un número comprendido entre 0 (correspondiente a 0 V) y 1023 (5 V).
7. Se invierte el valor del bit CHS0 para modificar el canal seleccionado (del canal 2 al 3 o viceversa). **CHS0 = ~CHS0**
8. Se esperan 0.1 s antes de volver al paso 1.

4.4.4 Configuración módulos CCP

Para cada uno de los servomotores se debe generar una señal PWM de período 20 ms (f = 50 Hz) y ciclo de trabajo entre 1 y 2 ms (de forma aproximada, ya que como se ha comentado antes, se ha modificado el rango del ciclo de trabajo, haciéndolo mayor).

La primera opción que puede pensarse en utilizar es el modo PWM de los módulos CCP. No obstante, este emplea el timer TMR2 (de 8 bits), y el máximo período que se puede programar es de 4.096 ms, frente a los 20 ms que se necesitan.

La solución pasa por recurrir al modo Comparación de los módulos CCP, ya que este utiliza el timer TMR1 (de 16 bits). Con TMR1 se temporizan 20 ms. Durante T_{ON}, la salida del comparador está a 1. Cuando TMR1 alcanza el valor almacenado en el registro de 16 bits CCPRxH:CCPRxL (correspondiente a la temporización de T_{ON}), la salida pasa a estar a 0, situación en la que permanece hasta que transcurren los 20 ms.

El problema que presenta el modo Comparación es que el funcionamiento ideado sólo es válido para un período y se desea que la señal PWM se esté enviando continuamente al servomotor. Para solventar este inconveniente, se habilita la interrupción asociada a TMR1.

La interrupción salta una vez transcurridos los 20 ms temporizados. En este momento, hay que resetear los módulos CCP y volver a configurarlos en modo Comparación para generar de nuevo un período de la señal PWM.

GIE = 1; PEIE = 1; ⇒ Int. global e interrupciones por periféricos habilitadas.
TMR1IE = 1; ⇒ Interrupción TMR1 habilitada.

Gestión interrupción TMR1

1. Se apaga TMR1. **TMR1ON = 0**
2. Se resetean los módulos CCP. **CCP1CON = 0x00; CCP2CON = 0x00**
3. Se reconfigura TMR1 para período de 20 ms.

$$\mathbf{TMR1H = 0xB1; TMR1L = 0xE0}$$

$$\mathbf{TMR1 = TMR1H:TMR1L = 256 * TMR1H + TMR1L =}$$

$$= 256 * (16*11 + 1) + (16*14 + 0) = 45536$$

Temporización TMR1: $T_d = (65536 - TMR1) * P * (4/F_{osc}) =$

$$= (65536 - 45536) * 1 * (4/4MHz) = \mathbf{20\ ms}$$

(P = 1, ya que se trata del valor por defecto y no se ha modificado)
4. Se introducen en CCPRxH y en CCPRxL los valores necesarios para conseguir T_{ON} correspondiente a la posición deseada por el usuario.

$$T_{ON} [\mu s] = CCPRxH:CCPRxL - TMR1 \Rightarrow$$

$$\Rightarrow CCPRxH:CCPRxL = TMR1 + 1000 * T_{ON} [ms]$$

$$CCPRxH:CCPRxL = 45536 + 1000 * T_{ON} [ms]$$

$$\mathbf{CCPRxH = (45536 + 1000 * T_{ON} [ms]) / 256}$$

$$\mathbf{CCPRxL = (45536 + 1000 * T_{ON} [ms]) \% 256}$$

Tener en cuenta que el valor de T_{ON} no tiene por qué ser necesariamente el mismo para el primer servomotor (que utiliza el módulo CCP1) que para el segundo (módulo CCP2).
5. Módulos CCP en modo comparación, con terminal CCPx iniciado a '1' y que se pone a '0' cuando TMR1 alcanza el valor del registro CCPRxH:CCPRxL.
CCP1CON = 0x09; CCP2CON = 0x09
6. Se baja el flag (**TMR1IF = 0**) y se enciende TMR1 (**TMR1ON = 1**).

4.5 Funcionamiento básico

Finalmente, tras haber explicado cómo se configuran los módulos se va a proceder a resumir cómo funciona el proyecto en su conjunto, es decir, cómo consigue el usuario que el robot alcance una posición determinada.

φ (giro alrededor del eje Z) y ψ (giro alrededor del eje X) del robot pueden ir aproximadamente de -90° a $+90^\circ$. Este intervalo se ha dividido para ambos ángulos en 11 posiciones angulares, numeradas del 0 al 10, luego existe una separación de 18° entre 2 posiciones consecutivas.

La posición 10 se corresponde con el menor T_{ON} (0.6 ms y 0.7 ms respectivamente para los servomotores empleados en este proyecto) y la posición 0 con el mayor T_{ON} (2.4 ms para ambos servomotores). Obviamente, T_{ON} para las posiciones de la 1 a la 9 procede de dividir el rango de T_{ON} del servomotor correspondiente en intervalos idénticos.

Los valores de CCPxH y CCPxL correspondientes a cada T_{ON} se calculan a partir de las expresiones mostradas en la configuración de los módulos CCP. Recordar que CCP1 está asociado al primer servomotor (giro alrededor del eje Z) y CCP2 al segundo (giro alrededor del eje X).

La posición 5 (0°) es la posición de reset y cada vez que el circuito se apaga ambos servomotores retornan a esta posición, independientemente de la posición que tuviesen antes de que se apagara el circuito.

La relación entre la posición angular de los servomotores y el joystick viene dada por:

- **Posición X del Joystick** controla el **giro (ángulo φ , giro en torno al eje Z)**
 - Resultado CAD: **0** → **-2 posiciones** para φ (gran giro hacia la izquierda)
 - Resultado CAD: **1-399** → **-1 posición** para φ (pequeño giro hacia la izquierda)
 - Resultado CAD: **400-600** → **NO** hay cambio en φ
 - Resultado CAD: **601-1022** → **+1 posición** para φ (pequeño giro hacia la der.)
 - Resultado CAD: **1023** → **+2 posiciones** para φ (gran giro hacia la derecha)
- **Posición Y del Joystick** controla la **desviación (ángulo ψ , giro en torno al eje X)**
 - Resultado CAD: **0** → **-2 posiciones** para ψ (gran giro hacia abajo)
 - Resultado CAD: **1-399** → **-1 posición** para ψ (pequeño giro hacia abajo)
 - Resultado CAD: **400-600** → **NO** hay cambio en ψ
 - Resultado CAD: **601-1022** → **+1 posición** para ψ (pequeño giro hacia arriba)
 - Resultado CAD: **1023** → **+2 posiciones** para ψ (gran giro hacia arriba)

Los intervalos de resultados de la conversión A/D han sido escogidos considerando que es muy fácil lograr que el joystick alcance una posición extrema (0 ó 1023) y que se requiere una mayor habilidad para situarlo en una posición intermedia (rango 1-399 ó 601-1023). Con el joystick en reposo (tensión de 2.5 V, resultado CAD de 512 o cercano), tampoco se modifica la posición de los servomotores.

Las posiciones angulares de los 2 servomotores pueden ser modificadas al mismo tiempo (en realidad hay una diferencia de 0.1 s) pero se debe recordar que cuando un

servomotora alcanza una posición extrema las órdenes del joystick serán ignoradas hasta que cambie la dirección de movimiento.

Nuevamente, volver a mencionar que presionar el pulsador del joystick sirve para activar el láser, cuyo funcionamiento es independiente del giro de los servomotores.

La imagen inferior trata de condensar el funcionamiento descrito.

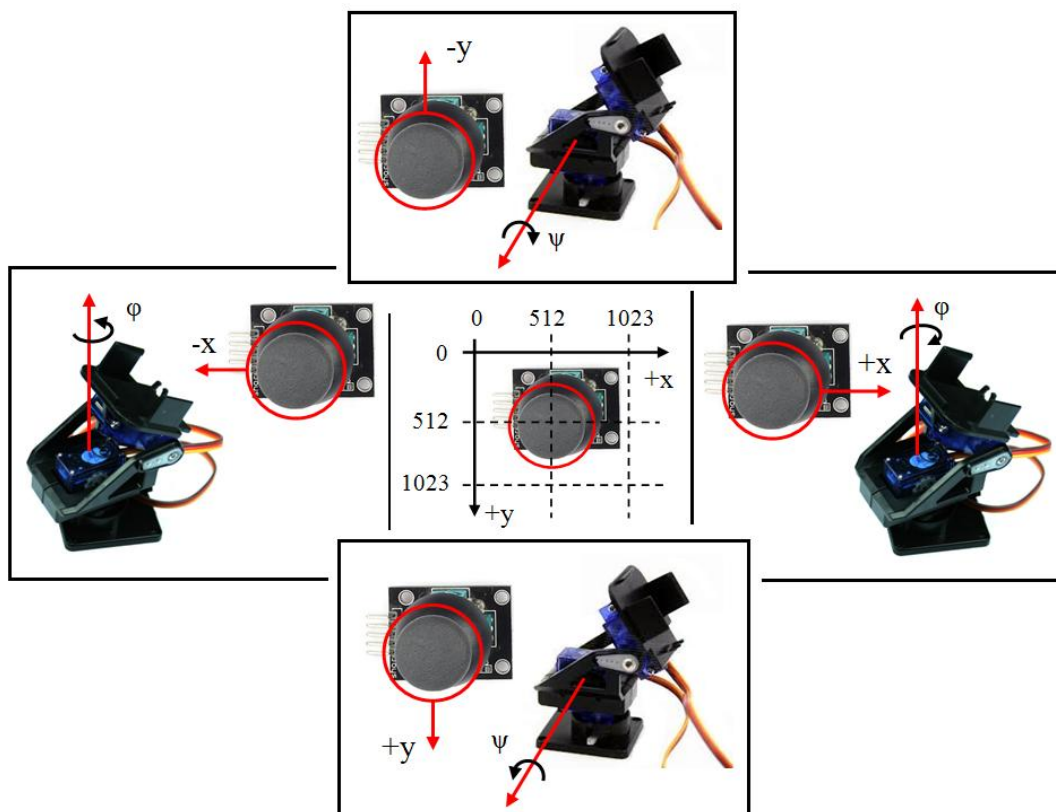


Figura 33. Funcionamiento Laser-shooting Robot

4.5.1 Diagramas de flujo

Finalmente, se recogen varios diagramas de flujo que sintetizan el funcionamiento del programa desde el punto de vista del microcontrolador (programa principal, interrupciones, funciones, módulos, timers,...). Notar que el funcionamiento del programa recae casi de forma exclusiva en las interrupciones.

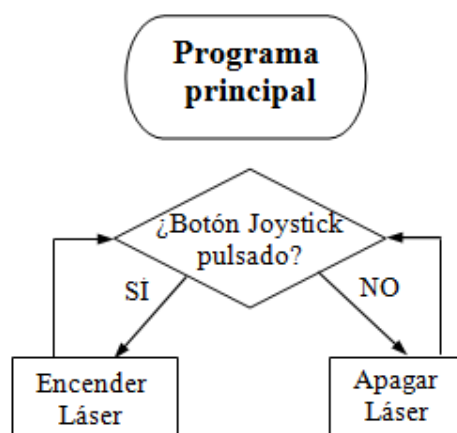


Figura 34. Diagrama de flujo. Programa principal (Laser-shooting robot).

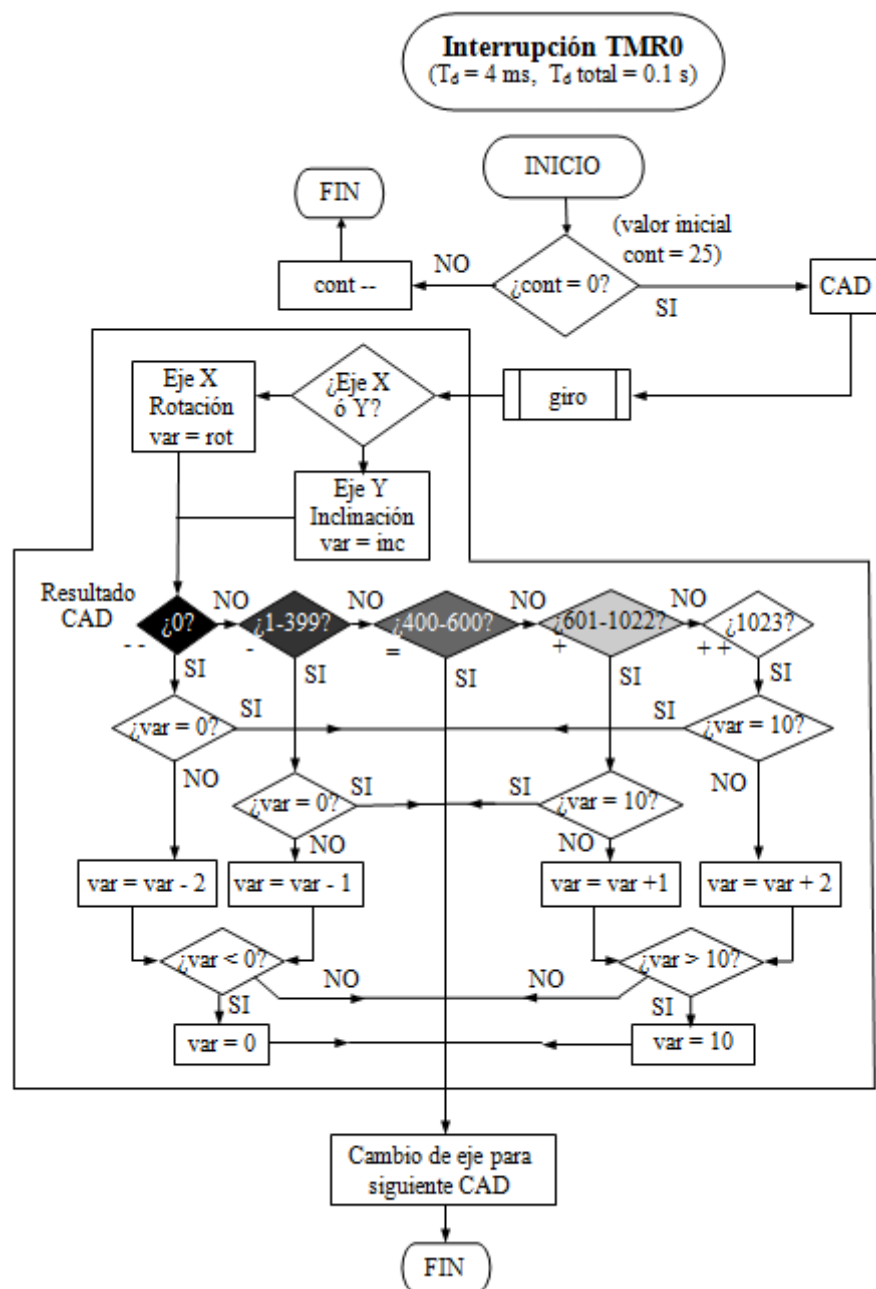


Figura 35. Diagrama de flujo. Interrupción TMR0 (Laser-shooting robot).

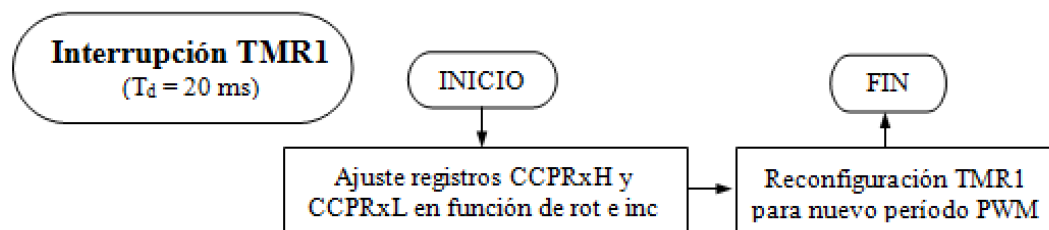


Figura 36. Diagrama de flujo. Interrupción TMR1 (Laser-shooting robot)

4.6 Programa fuente

Se recoge a continuación el programa fuente. Mencionar que la función que gestiona el resultado de la conversión A/D, llamada *giro()*, dispone de una parte comentada que sirve para mostrar el resultado de la conversión A/D en el display LCD de la placa PICDEM 2 PLUS, con el valor correspondiente al eje X del joystick en la primera línea y el valor correspondiente al eje Y en la segunda.

Se ha mantenido este código porque resultó útil en la fase de pruebas realizada con la placa PICDEM 2 PLUS. Si se quiere emplear, tener en cuenta que hay que incluir la librería del display LCD e inicializarlo (las correspondientes líneas de código ya están, sólo hay que quitar los comentarios).

```
#include<htc.h> //Se incluye la librería del micro a usar.
//#include "lcd.h" //Se incluye la librería de la pantalla LCD
#include "stdio.h"

#define XTAL_FREQ 4000000 //Oscilador interno de 4MHZ.
CONFIG(WRT OFF & WDTE OFF & PWRTE OFF & FOSC XT & LVP OFF & BOREN OFF);
//BOREN OFF: se desactiva el Brown-out Reset, es decir, cuando se produzca una
//caída en la tensión de alimentación, el programa no se reseteará al recuperar el
//nivel adecuado de alimentación; seguirá desde el punto en el que se encontraba.

unsigned char contTMR0=25; //Contador asociado a TMR0.
unsigned char rot=5, inc=5; //Variables para la rotación y la inclinación.

//Arrays de constantes con los valores a introducir en los registros CCP para cada
//uno de los posibles valores de la rotación (eje x) y la inclinación (eje y).
const unsigned char roth[11]={0xBB,0xBA,0xB9,0xB9,0xB8,0xB7,0xB7,0xB6,0xB5,0xB4,0xB4};
const unsigned char rotL[11]={0x40,0x8C,0xD8,0x24,0x70,0xBC,0x08,0x54,0xA0,0xEC,0x38};
const unsigned char inch[11]={0xBB,0xBA,0xB9,0xB9,0xB8,0xB7,0xB7,0xB6,0xB5,0xB5,0xB4};
const unsigned char incL[11]={0x40,0x96,0xEC,0x42,0x98,0xEE,0x44,0x9A,0xF0,0x46,0x9C};

void giro(unsigned char CH); //Función para actualizar rot (controlada con eje x del
//joystick) e inc (controlada con eje y del joystick).

//Interrupciones
static void interrupt isr(void){

    if(T0IF){ //INTERRUPCIÓN TMR0
        T0IF=0; TMR0=0x06; //Se baja el flag de TMR0 y se reinicia TMR0 a 0x06.
        contTMR0--; //Se decreuenta el contador asociado a TMR0.
        if(contTMR0==0){ //contTMR0 = 0, temporización total: 25*4 = 100ms = 0.1s
            ADON=1; //Se activa el conversor A/D.
            delay us(25); //Tiempo de espera para que concluya la adquisición.
            GO_DONE=1; //Se inicia la conversión.
            contTMR0=25; //Se reinicia contTMR0 a 25.
        }
    }

    if(TMR1IF){ //INTERRUPCIÓN TMR1
        TMR1ON=0; //Se apaga TMR1.
        CCP1CON=0x00; CCP2CON=0x00; //Se apagan CCP1 y CCP2.
        CCP1H=roth[rot]; //Se actualiza el valor de CCP1H y CCP1L
        CCP1L=rotL[rot]; //en función del valor de rot.
        CCP2H=inch[inc]; //Se actualiza el valor de CCP2H y CCP2L
        CCP2L=incL[inc]; //en función del valor de inc.
        TMR1H=0xB1; TMR1L=0xE0; //TMR1 se reinicia a 45536.
        CCP1CON=0x09; CCP2CON=0x09; //CCP1 y CCP2: modo comparación, paso de '1' a '0'.
        TMR1IF=0; TMR1ON=1; //Se baja el flag de TMR1 y se enciende de nuevo.
    }

    if(ADIF){ //INTERRUPCIÓN CONVERSADOR A/D
        ADIF=0; ADON=0; //Se baja el flag y se apaga el conversor A/D.
        giro(CH); //Se actualiza rot(CH=0) o inc(CH=1).
        CHS0=~CHS0; //Se cambia el canal del CAD seleccionado
    }
}; //del 2 al 3 o viceversa.
```

```

    }

//Programa principal
void main(void){
    //lcd init();           //Inicialización pantalla LCD.

    ADCON1=0x80;           //ADFM=1, ajuste a la derecha para CAD; ADCS2=0.
                           //RA0-RA3, RA5 y RE0-RE2; terminales analógicos.
    TRISA=0xFF;            //PORTA: terminales configurados como entradas.
    TRISC=0x00;            //PORTC: terminales configurados como salidas.

    OPTION_REG=0x03;       //TMR0 como temporizador con predivisor 16.
    TMR0=0x06;             //Td = (256-6)*16*1us = 4 ms
    ADCON0=0x50;           //ADCS1:ADCS0=01 (ADCS2=0), f=fosc/8 -> TAD = 2us > 1.6us
                           //CHS2:CHS0=010, canal 2 seleccionado.
                           //ADON=0, módulo A/D apagado.
    GIE=1; PEIE=1;         //Habilitación interrupciones globales y por periféricos.
    T0IF=0; T0IE=1;        //Se baja el flag de TMR0 y se habilita su interrupción.
    ADIF=0; ADIE=1;        //Se baja el flag de CAD y se habilita su interrupción.

    TMR1H=0xB1; TMR1L=0xE0; //TMR1 se inicia a 45536.
    TMR1IF=0; TMR1IE=1;     //Se baja el flag de TMR1 y se habilita su interrupción.
    T1CON=0x01;             //Reloj interno, predivisor = 1, TMR1 encendido.
                           //Temporización TMR1: (65536-45536) · 1 · 1us = 20 ms

    while(1){               //Bucle infinito.
        if(~RA4){RC0=1;}    //Pulsación botón del joystick, láser encendido.
        else{RC0=0;}       //Botón joystick no pulsado, láser apagado.
    }

void giro(unsigned char CH){
    unsigned int eje;        //Variable que recupera el valor de CAD.
    signed char g, var;      //g establece la variación del valor de rot ó inc.
                           //var desempeña el papel de rot ó inc.

    eje=256*ADRESH+ADRESL;  //Se calcula la posición x ó y del joystick.
    if((400<=eje)&&(eje<=600)){g=0;} //400<=eje<=600, joystick en reposo (g=0).
    else if(eje==0){g=-2;} //eje=0, valor extremo sentido - (g=-2).
    else if((0<eje)&&(eje<400)){g=-1;} //0<eje<400, valor medio sentido - (g=-1).
    else if((600<eje)&&(eje<1023)){g=1;} //600<eje<1023, valor medio sentido + (g=1).
    else if(eje==1023){g=2;} //eje=1023, valor extremo sentido + (g=2).

    if(g!=0){               //g distinto de 0, se producirá movimiento.
        if(CH==0){var=rot;} //CH=0 (canal 2, eje x), variable a modificar es rot.
        else{var=inc;}     //CH=1 (canal 3, eje y), variable a modificar es inc.
        if(~((var==10 && g>0)|| //Excepciones: valor máximo y movimiento en sentido +
            (var==0 && g<0))){ //o valor mínimo y movimiento en sentido -.
            var=var+g;      //Se cambia el valor de var sumándole g.
            if(var>10){var=10;} //Si var supera el máximo, var toma valor máximo(10).
            else if(var<0){var=0;} //Si var no llega al mínimo, var toma valor mínimo(0).
            if(CH==0){rot=var;} //CH=0 (canal 2), variable modificada se copia en rot.
            else{inc=var;}     //CH=1 (canal 3), variable modificada se copia en inc.
        }
    }

    //Escritura del valor eje en la pantalla LCD.
    //if(CH==0){lcd goto(0x00);} //CH=0 (canal 2, eje x), se escribe en la 1ª línea.
    // else{lcd goto(0x40);}     //CH=1 (canal 3, eje y), se escribe en la 2ª línea.
    //lcd_putch(eje/1000+0x30); //Escritura de las unidades de millar en la LCD.
    //lcd_putch((eje/100)%10+0x30); //Escritura de las centenas en la pantalla LCD.
    //lcd_putch((eje/10)%10+0x30); //Escritura de las decenas en la pantalla LCD.
    //lcd_putch(eje%10+0x30); //Escritura de las unidades en la pantalla LCD.
}

```

4.7 PCB

Se muestran bajo estas líneas imágenes en 2D y 3D del PCB diseñado. Las referencias de los componentes se pueden consultar en la **Tabla 17**.

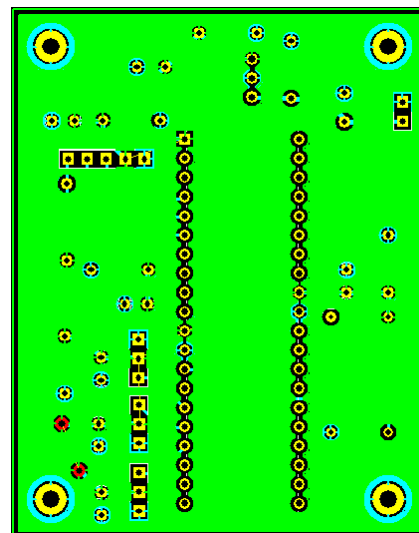
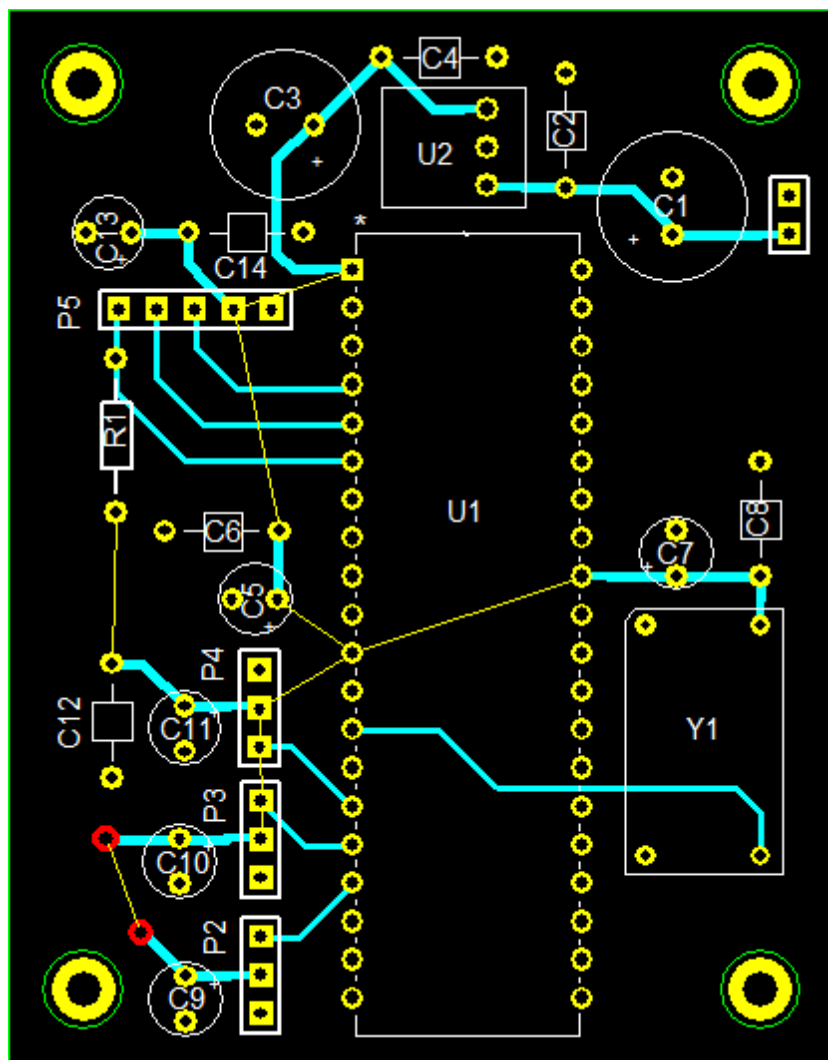


Figura 37. Vista del PCB con planos a escala 1/1 (arriba) y vista del PCB sin planos a escala 2/1 (izq.).

El punto rojo permite identificar las esquinas correspondientes de ambas vistas del PCB.

La vista de la cara inferior se obtiene dando la vuelta al PCB por el lado más largo más alejado del oscilador.

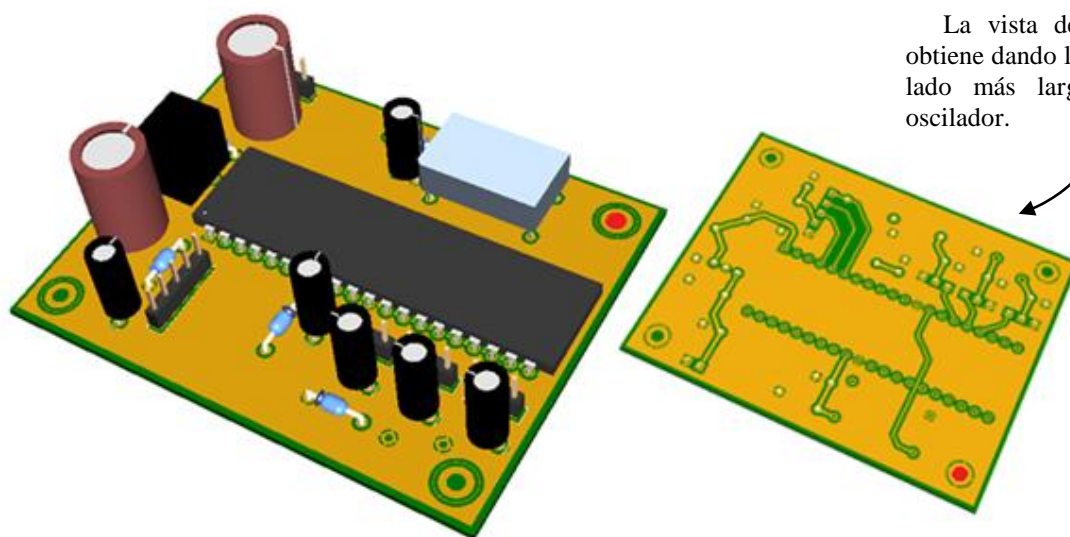


Figura 38. Vista 3D del PCB (cara superior e inferior)

4.8 Diseño 3D

El diseño 3D realizado para este proyecto cuenta con 4 partes principales: caja, tapa, soporte para el “robot” (estructura con los 2 servomotores) y soporte para el joystick.

- **Caja:** tiene 4 pequeñas columnas interiores donde encajar el PCB (se deben utilizar tornillos de métrica M2 o inferior) y 4 columnas exteriores de mayores dimensiones para sujetar la tapa. El conector del transformador de 9 V se encaja en un agujero situado junto a una de las esquinas.

La caja también tiene un compartimento para la batería, aunque finalmente se ha decidido cambiar la fuente de alimentación y este compartimento ya no se necesita. No obstante, se ha mantenido por si posteriormente se encuentra una solución a este problema.

- **Tapa:** dispone de 2 agujeros circulares grandes, uno en la parte posterior para el joystick y otro en la parte frontal para el robot; y 4 pequeños agujeros para los cables de los servomotores, los cables del láser y el interruptor. El agujero para el joystick debe ser lo suficientemente grande para garantizar su libertad de movimientos.
- **Soportes:** se ha diseñado un soporte para el robot y otro para el joystick. Por lo que respecta al montaje, la parte inferior de la tapa posee salientes que encajan en los agujeros de los soportes.

Tras ensamblar todas las piezas, el resultado final es:

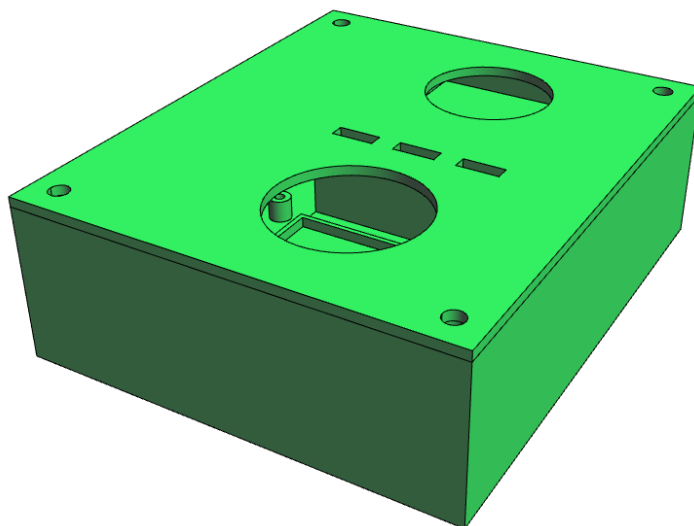


Figura 39. Montaje final Laser-shooting Robot

4.9 Multimedia

Se ha publicado un artículo en inglés sobre este proyecto bajo el título “**LASER-SHOOTING ROBOT- nerds’ enemies beware!**” en el sitio web DesignSpark, vinculado a la empresa RS Components.

En el artículo se incluyen todos los archivos vinculados al proyecto: código fuente, diseño del esquemático, diseño del PCB y diseño 3D.

- [Enlace al artículo](#)



Figura 40. Cabecera de presentación del artículo

5 Cubo de LEDs RGB 3x3x3

5.1 Introducción

El objetivo de este proyecto es diseñar un cubo de LEDs de dimensiones 3x3x3 (ancho x largo x alto) en el que se añade otro factor x3, ya que los LEDs empleados son RGB (Red, Green, Blue), aumentando el atractivo del mismo. Cada LED (81 en total) podrá ser encendido de forma independiente.

No se trata de un proyecto complejo desde el punto de vista de la programación, pero se requiere cierta originalidad para que el resultado final sea vistoso. Además, el montaje de la estructura de LEDs (se utilizan LEDs de agujero pasante) exige cierta habilidad y soltura a la hora de soldar, factor que contribuye a aumentar la dificultad. Por ello se va a hacer más hincapié en este aspecto que en los proyectos anteriores.



Figura 41. LED RGB

5.2 Diseño esquemático

Para el bloque de alimentación, que utiliza la batería de 9 V, y el microcontrolador consultar el apartado 2.1, con la salvedad de que en este proyecto, a diferencia de los anteriores, el regulador empleado es un regulador lineal de 5 V y 2 A, debido a la elevada corriente demandada por los LEDs RGB (60 mA como máximo cada uno).

Se dedica una sección al LED RGB debido a su importancia en este proyecto. Posteriormente se incluyen el esquemático y el listado de componentes.

5.2.1 LED RGB

A pesar de que un LED no posee la complejidad de un circuito integrado, hay que destacar una serie de factores que deben considerarse del proyecto:

- **Búsqueda del LED RGB:** cuando en la página web de un fabricante se busca un LED, entre los criterios utilizados para cribar toda la selección se encuentra el color del LED. Sin embargo, la opción RGB puede no aparecer o no contener todos los productos de este tipo, por lo que se debe ser cuidadoso cuando busca.

Por ejemplo, hay fabricantes que clasifican sus LEDs RGB bajo el color Rojo y otros crean varias categorías “repetidas” con distintos productos como Red, Green, Blue por un lado y Rojo, Verde, Azul por otro. El término “multicolor” también suele resultar bastante útil en las búsquedas.

- **Pines LED RGB:** es habitual que los LEDs RGB posean un terminal común a los 3 LEDs individuales. Es necesario identificarlo (en los LEDs de agujero pasante suele ser el terminal con la patilla más larga) y determinar si es el ánodo o el cátodo. Para los LEDs empleados se trata del ánodo.

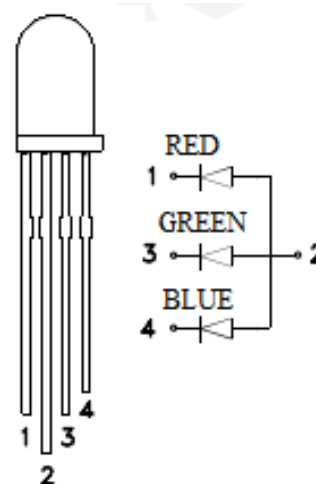


Figura 42. Patillaje y conexionado del LED RGB empleado.

El resto de terminales también suelen tener distintas longitudes para facilitar su identificación. No obstante, se debe prestar atención, porque a pesar de llamarse RGB, a veces los terminales están ordenados como RBG. No es el caso de los LEDs utilizados.

- **Colores:** teniendo en cuenta que los LEDs rojo, verde y azul del LED RGB se pueden encender de forma simultánea se pueden conseguir, además de los colores primarios, 4 colores más: magenta (rojo + azul), cian (azul + verde), amarillo (verde + rojo) y blanco (rojo + azul + verde).

Adicionalmente, modulando la intensidad (control de corriente) que circula por cada uno de los LEDs se puede recorrer todo el espectro cromático, aunque esta posibilidad no se abordará en este proyecto. En caso de querer implementar esta función se recomienda la utilización del circuito integrado TLC5940, que permite efectuar este control para 16 LEDs (para un número superior de LEDs, como ocurre en este caso, se puede conectar varios integrados en serie).

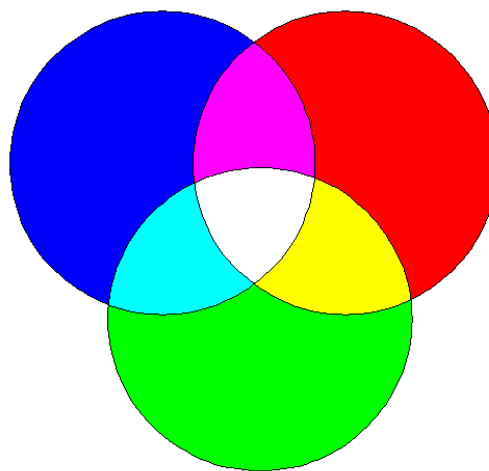


Figura 43. Colores que se pueden conseguir con un LED RGB con un control ON/OFF.

- **Resistencias:** los LEDs requieren una cierta corriente para encenderse (en este caso 20 mA para los 3 LEDs que integran el LED RGB). Considerando la tensión aplicada (5 V) y la caída de tensión en cada LED, que varía al estar relacionada con la longitud de onda (color), se pueden determinar las resistencias requeridas.

Las caídas de tensión típicas son de 1.95 V, 3.3 V y 3.3 V para los LEDs de color rojo, azul y verde respectivamente, por lo que se emplea una resistencia de 150 Ω para el LED rojo y resistencias de 82 Ω para los LEDs azul y verde.

- **Conexionado:** los 27 LEDs RGB se organizan en 3 niveles de 9 LEDs. Los ánodos de todos los LEDs de un mismo nivel se encuentran conectados entre sí, mientras que los cátodos de los LEDs de cada nivel se hallan conectados a los cátodos correspondientes de los LEDs de los restantes niveles.

La posibilidad de encendido de los LEDs de cada nivel se controla con un transistor bipolar PNP 2N3906 que posee una corriente de colector (I_C) máxima de 800 mA (la corriente máxima demandada por un nivel será de 540 mA, teniendo en cuenta que hay 27 LEDs individuales).

La base de cada transistor se encuentra conectada al microcontrolador a través de una resistencia de 1 k Ω y un diodo que garantiza que los transistores pasen a corte cuando hay un '1' lógico en el terminal correspondiente (RA5, RA4, RA3 para los transistores de los niveles 1, 2 y 3 respectivamente). Con un '0' lógico el transistor entra en saturación y el ánodo de los LEDs del nivel correspondiente pasa a estar a 5 V, lo que permite su encendido.

Sólo los cátodos de los LEDs del nivel inferior se encuentran conectados directamente a los pines del microcontrolador (se utilizan 9 resistencias de 150 Ω y

18 resistencias de $82\ \Omega$), pero dado que los cátodos de los LEDs de los diferentes niveles se encuentran conectados entre sí, es posible controlar los 27 LEDs.

No obstante, si se activa más de un nivel al mismo tiempo los colores de los LEDs han de ser necesariamente los mismos. En cualquier caso, esta situación se va a evitar, y se va a activar un único nivel cada vez para minimizar la corriente demandada por todo el conjunto (aunque el bloque de alimentación está dimensionado para poder encender los 3 niveles de forma simultánea).

Entonces, ¿se pueden encender varios niveles de forma simultánea con diferentes colores? Estrictamente no, pero la solución radica en conmutar tan rápido el nivel seleccionado que el ojo humano perciba que los 3 niveles están encendidos al mismo tiempo (se necesita una velocidad de refresco de entre 40 y 200 Hz).

5.2.2 Esquemático

Se recoge en esta página el esquemático del proyecto.

La batería y su conector no están representados en el esquemático al no ser elementos integrados en el PCB aunque sí que se indican los terminales a los que se conectan. El terminal con la leyenda (N.C) (pin “1” del oscilador) no se encuentra conectado.

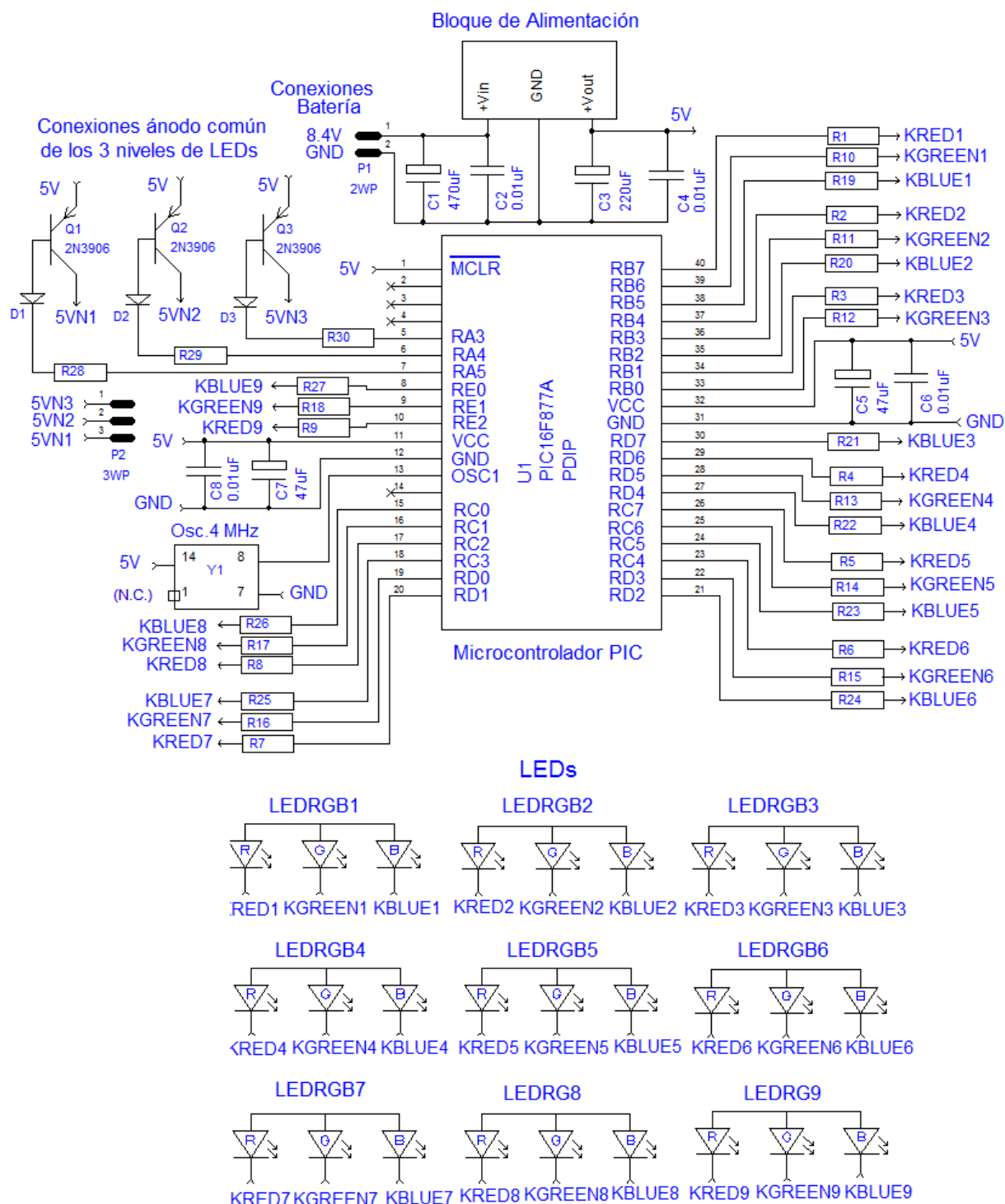


Figura 44. Esquemático (Cubo de LEDs RGB).

5.2.3 Listado de componentes

Se detallan a continuación todos los componentes utilizados en el proyecto. Para todos ellos se especifica, a excepción de la batería y su conector (elementos que no forman parte del PCB), su referencia, una breve descripción, el tipo de empaquetado y su cantidad.

Referencia	Descripción	Valor	Empaquetado	Uds
P1	Conector alimentación (2 terminales)		DSC	1
P2	Conector (3 terminales)		DSC	1
R1-9	Resistencia LED Rojo	150 Ω	DSC	9
R10-27	Resistencia LED Verde/Azul	82 Ω	DSC	18
R28-30	Resistencia de base	1 k Ω	DSC	3
D1-3	Diodo 1N4148		SOD27	3
C1	C _{IN} regulador	470 μ F	DSCV	1
C3	C _{OUT} regulador	220 μ F	DSCV	1
C2, C4, C6, C8	Condensador de desacoplo	0.01 μ F	DSC	4
C5, C7	Condensador de reserva	47 μ F	DSCV	2
LED1-27	LED RGB de ánodo común		DSC	27
Q1-3	Transistor bipolar PNP BC327		TO-92	3
U1	PIC16F877A		PDIP	1
U2	Regulador L78S05CV		TO-220	1
Y1	Oscilador f = 4MHz		DIP	1
-	Batería 9 V			1
-	Conector batería			1

Tabla 19. Listado de componentes proyecto Cubo de LEDs RGB.

5.3 Configuración del microcontrolador

En este proyecto no se utiliza ningún módulo del microcontrolador ni tan siquiera entradas, únicamente salidas digitales: 3 para controlar el estado de los transistores (corte/saturación) que regulan el encendido de cada nivel y 27 para los LEDs (3 cátodos por 9 LEDs RGB). Solamente será necesario configurar los puertos. Curiosamente, al mismo tiempo se trata del proyecto que usa un mayor número de pines del microcontrolador.

5.3.1 Configuración de los puertos

Los terminales del microcontrolador utilizados son:

CÁTODOS LEDS RGB								
Pin	Color	Nº LED RGB	Pin	Color	Nº LED RGB	Pin	Color	Nº LED RGB
<i>RB7</i>	R	1	<i>RB4</i>	R	2	<i>RB1</i>	R	3
<i>RB6</i>	G	1	<i>RB3</i>	G	2	<i>RB0</i>	G	3
<i>RB5</i>	B	1	<i>RB2</i>	B	2	<i>RD7</i>	B	3
Pin	Color	Nº LED RGB	Pin	Color	Nº LED RGB	Pin	Color	Nº LED RGB
<i>RD6</i>	R	4	<i>RC7</i>	R	5	<i>RC4</i>	R	6
<i>RD5</i>	G	4	<i>RC6</i>	G	5	<i>RD3</i>	G	6
<i>RD4</i>	B	4	<i>RC5</i>	B	5	<i>RD2</i>	B	6
Pin	Color	Nº LED RGB	Pin	Color	Nº LED RGB	Pin	Color	Nº LED RGB
<i>RD1</i>	R	7	<i>RC2</i>	R	8	<i>RE2</i>	R	9
<i>RD0</i>	G	7	<i>RC1</i>	G	8	<i>RE1</i>	G	9
<i>RC3</i>	B	7	<i>RC0</i>	B	8	<i>RE0</i>	B	9
TERMINALES DE CONTROL TRANSISTORES								
<i>RA5</i>	Transistor Nivel 1		<i>RA4</i>	Transistor Nivel 2		<i>RA3</i>	Transistor Nivel 3	

Tabla 20. Salidas del microcontrolador (Cubo de LEDs RGB)

Por tanto, los puertos se configuran de la siguiente forma:

Registros

ADCON1 = 0x06; \Rightarrow PCFG = 0b0000 \Rightarrow

\Rightarrow *RA0-RA3*, *RA5* y *RE0-RE2*; terminales digitales.

(*RA4* es el único terminal de PORTA siempre digital)

TRISA = 0x00; TRISB = 0x00; TRISC = 0x00; TRISD = 0x00; TRISE = 0x00; \Rightarrow

\Rightarrow Todos los terminales PORTA, PORTB, PORTC, PORTD y PORTE configurados como salidas.

5.4 Programación

El propósito perseguido es programar el cubo de LEDs para que se ilumine siguiendo una secuencia. En el momento de la redacción de esta memoria todavía no se ha finalizado la programación. No obstante, sí que se han programado una serie de conceptos básicos que se explican a continuación.

➤ ON/OFF

Es interesante notar que:

'0' lógico en la base de un transistor \Rightarrow Nivel correspondiente **ON** (5 V en ánodo)

'1' lógico en la base de un transistor \Rightarrow Nivel correspondiente **OFF** (0 V en ánodo)

Además suponiendo que el nivel correspondiente a un cierto LED esté encendido:

'0' lógico en el cátodo del LED \Rightarrow LED **ON**

'1' lógico en el cátodo del LED \Rightarrow LED **OFF**

Puede apreciarse que en ambos casos la lógica es negativa, es decir, **ON** se corresponde con el **'0' lógico** mientras que **OFF** se corresponde con el **'1' lógico**. Como suele resultar más sencillo y cómodo programar con lógica positiva, se van a utilizar dos directivas *#define* para asociar **ON** a **0** y **OFF** a **1**. Cada vez que se escriba **ON** el compilador lo sustituirá por **0** y cada vez que se escriba **OFF** el compilador lo reemplazará por **1**.

Por otro lado, este análisis sirve para apreciar que, en el peor de los casos (0 V en el ánodo y 5 V en el cátodo), un LED debe soportar una tensión inversa de 5 V. No obstante, los tres LEDs (rojo, verde, azul) son capaces de soportar esta tensión en inversa (dato recogido en la hoja de características y comprobado de forma experimental). En cualquier caso, se va a tratar de evitar esta situación, procurando que siempre que el ánodo esté a 0 V (nivel apagado), el cátodo también lo esté.

➤ Color del LED

También se han utilizado directivas para simplificar la gestión del color con el que se desea iluminar cada LED RGB. Se han creado funciones que a partir del color deseado, determinan qué LEDs (rojo, verde, azul) deben estar encendidos.

El número que se asocia a cada color mediante *#define*, viene dado por:

Color	Cátodo R	Cátodo G	Cátodo B	Nº
Blanco / White (W)	0	0	0	0
Amarillo / Yellow (Y)	0	0	1	1
Magenta / Magenta (M)	0	1	0	2
Rojo / Red (R)	0	1	1	3
Cian / Cyan (C)	1	0	0	4
Verde / Green (G)	1	0	1	5
Azul / Blue (B)	1	1	0	6
LED apagado (LEDOFF)	1	1	1	7

Tabla 21. Número asociado a cada color.

A partir del número es muy fácil calcular mediante divisiones, utilizando restos y cocientes, qué LEDs tienen que estar encendidos para conseguir el color correspondiente. Nuevamente se ha recurrido al uso de directivas porque resulta más cómodo de programar y de entender por ejemplo, que la letra 'R' representa el color rojo, que vincular mentalmente el número '3' al color rojo.

➤ *Funciones*

Se han creado 2 funciones:

- **colourLED()**: pasándole un array con 9 colores y un número entre 1 y 3, esta función se encarga de iluminar los LEDs del nivel correspondiente según los colores pedidos y después apagarlos.
- **colourcube()**: ilumina todo el cubo, llamando 3 veces consecutivas a la función anterior, 1 para cada nivel, comenzando por el inferior. Sus entradas son un array de 27 posiciones y un parámetro (x).

Si $x = 0$, los 27 LEDs se iluminan según los colores indicados en el array. Si $x = 1$, significa que todos los LEDs de un mismo nivel se quieren iluminar del mismo color. El color en la posición 0 del array será el del primer nivel, el color en la posición 1; el del segundo nivel, y el color en la posición 2; el del tercer nivel. Si $x = 2$, quiere decir que se desean iluminar todos los LEDs con el mismo color, el indicado en la posición 0 del array.

Recordar que el tiempo que transcurre entre el encendido de los distintos niveles debe ser lo suficientemente breve como para crear el efecto de que todo el cubo está encendido. Todavía se está calibrando este factor.

Por otro lado, tener en cuenta que el orden de numeración de los LEDs es el recogido en el esquemático (**Figura 44**), que también se ha intentado reflejar en la **Tabla 20**; si bien donde se puede apreciar con mayor claridad es en el PCB (**Figura 45**), donde está representada la huella PCB (*footprint*) de los componentes. Esta numeración se respeta para los 3 niveles de LEDs.

5.5 PCB

Se recogen en este apartado las imágenes en 2D y 3D del PCB diseñado. Las referencias de los componentes se pueden consultar en la **Tabla 19**.

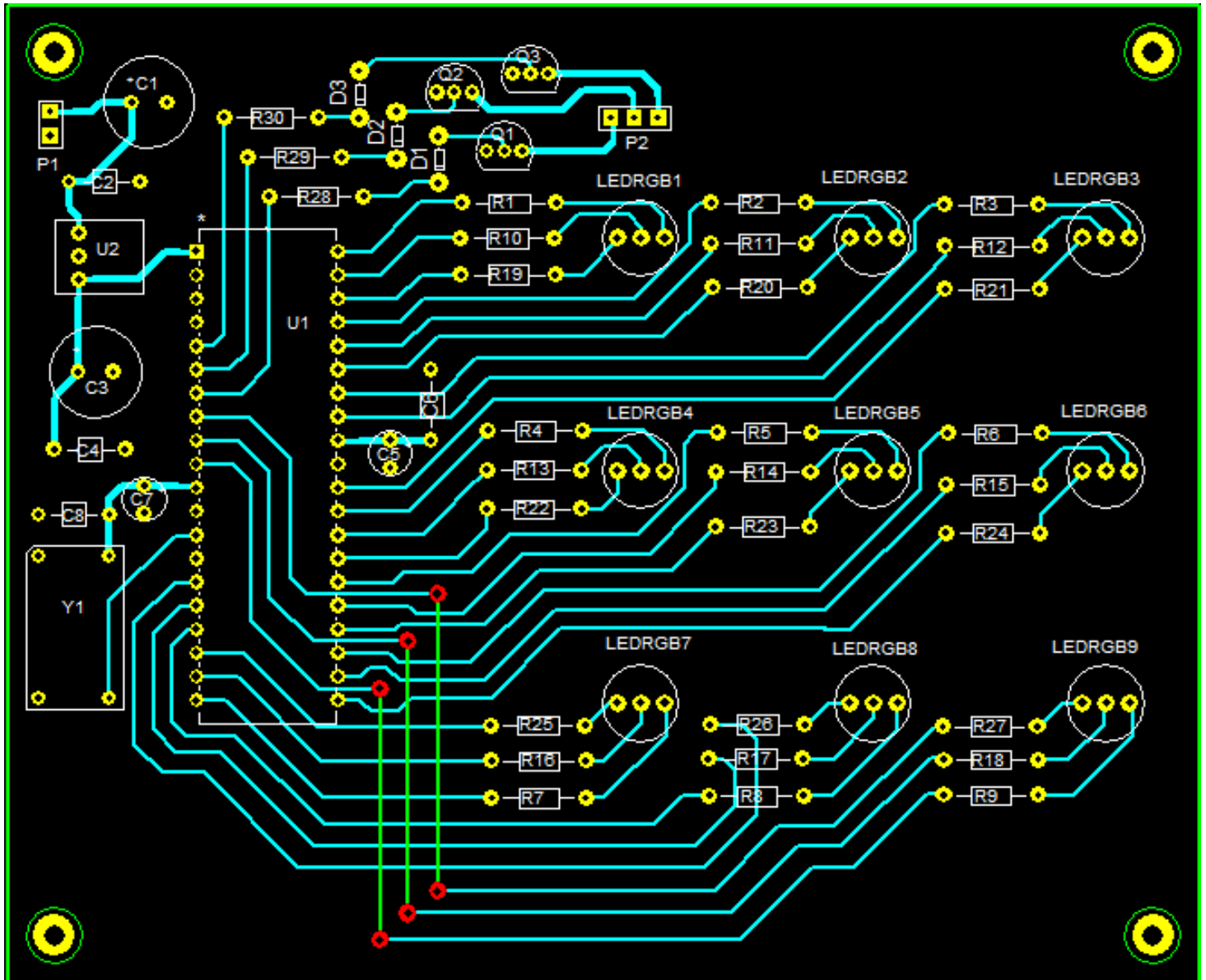


Figura 45. Vista del PCB sin planos a escala 3/2.

Nuevamente, se vuelve a destacar la importancia de observar el orden en el que se han numerado los LEDs (de izquierda a derecha, de arriba abajo).

Respecto al diseño del PCB, cabe señalar que las pistas situadas en la capa superior son perfectamente perpendiculares a las pistas de la capa inferior, con lo que se logra minimizar el crosstalk entre ambas.

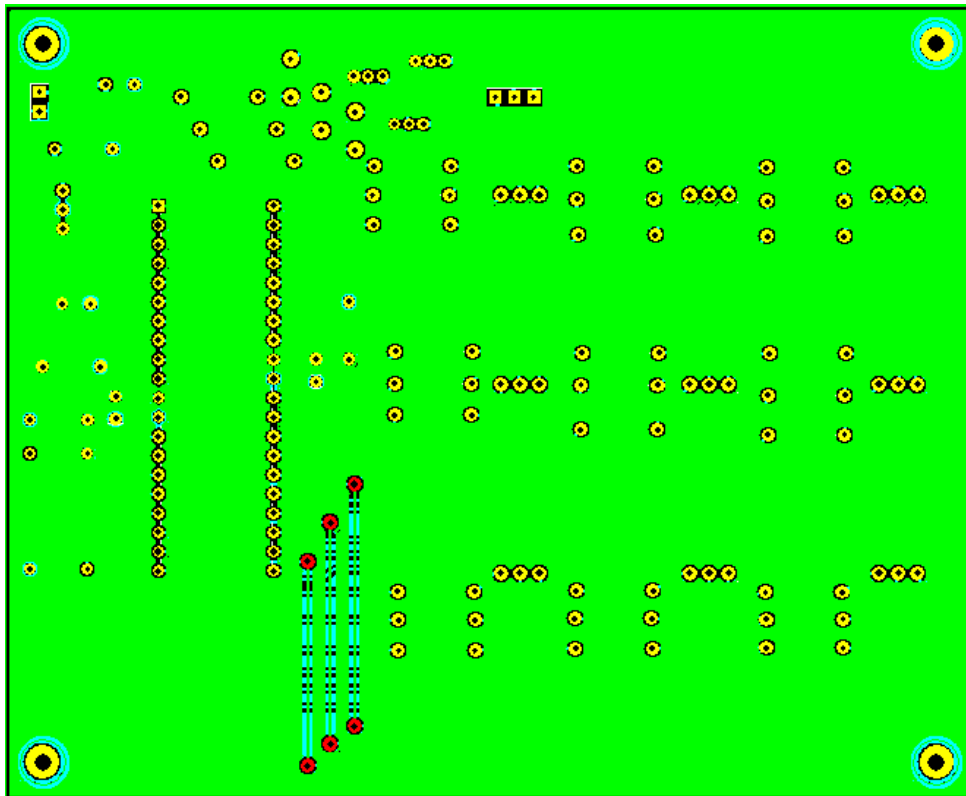
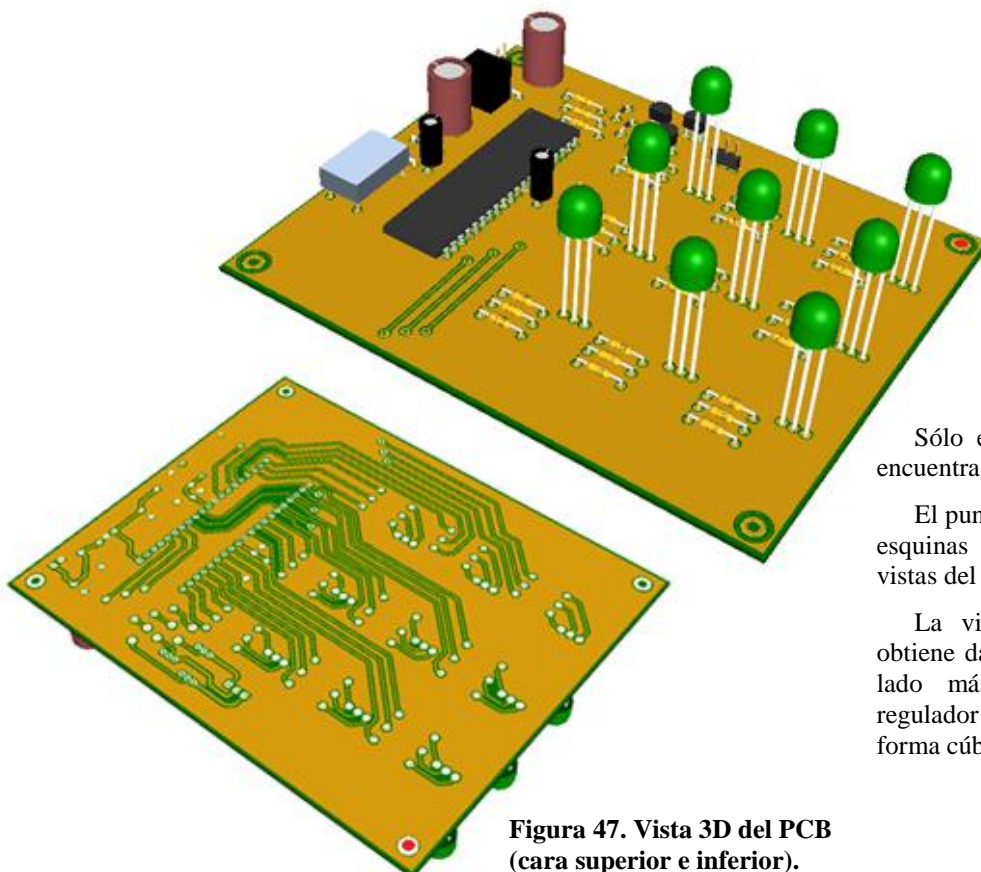


Figura 46. Vista del PCB con planos a escala 1/1.



Sólo el primer nivel de LEDs se encuentra representado en la vista 3D.

El punto rojo permite identificar las esquinas correspondientes de ambas vistas del PCB.

La vista de la cara inferior se obtiene dando la vuelta al PCB por el lado más largo más alejado del regulador (empaquetado negro con forma cúbica).

**Figura 47. Vista 3D del PCB
(cara superior e inferior).**

5.6 Montaje

El proceso de montaje del cubo de LEDs consta de los siguientes pasos:

- 1) Se cogen 9 LEDs y se colocan boca abajo formando un cuadro de dimensiones 3x3, manteniéndose los LEDs equidistantes entre sí a una distancia igual o ligeramente inferior a la longitud de la patilla del terminal común (en este caso, el ánodo, de unos 2.5 cm de longitud). Es conveniente que las patillas de los LEDs no estén alineadas con ninguno de los lados del cuadrado, ya que esto dificulta el proceso de soldadura (ver imagen a la derecha).

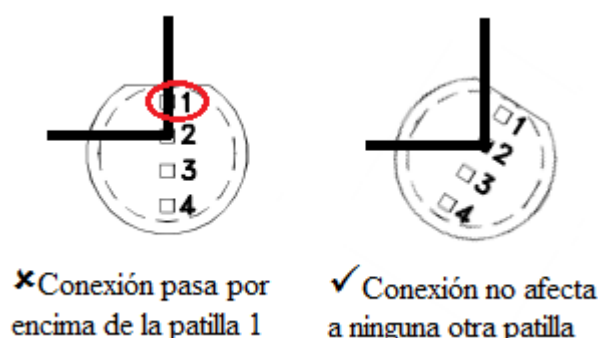


Figura 48. Como NO y como SÍ se debe posicionar el LED.

Para que los LEDs permanezcan en esta posición sin caerse se debe disponer de una superficie en la que previamente se hayan perforado agujeros donde introducir las cabezas de los LEDs o simplemente emplear una superficie lisa y algún adhesivo.

- 2) Se procede a soldar los ánodos de todos los LEDs del mismo nivel entre sí. Para ello se dobla la patilla del ánodo hasta que esta quede horizontal. Debe llegar al inicio de la patilla del ánodo del siguiente LED.

Tras cerrar todo el cuadrado exterior, se suelda el ánodo del LED central a cualquiera de los ánodos de los LEDs exteriores. Finalmente se comprueba la continuidad para verificar que todas las soldaduras están bien realizadas. Se dispone en este momento de un cuadrado de 3x3 LEDs de uno 5x5 cm.

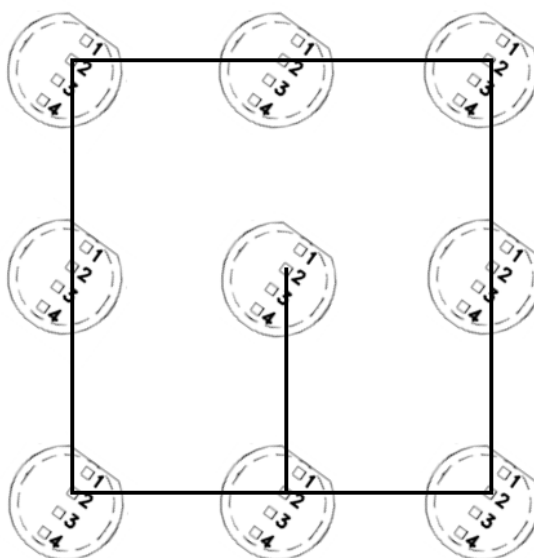


Figura 49. Vista en planta de un nivel soldado.

- 3) Se repiten los pasos 1 y 2 con los otros dos niveles de LEDs.

- 4) El siguiente paso es soldar los 3 niveles de LEDs. Pero antes de empezar conviene doblar ligeramente con unos alicates el inicio y el final de las patillas de cada uno de los cátodos de 2 de los 3 niveles. El objetivo es facilitar el proceso de soldadura y reducir la posibilidad de soldar cátodos de distintos colores entre sí por encontrarse muy juntos.

El nivel al que se le dejan las patillas rectas es el inferior (nivel 1) y por tanto, el último en ser soldado al conjunto ya que se empieza por el superior (nivel 3).

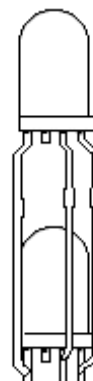


Figura 50. Forma recomendada de doblar las patillas.

- 5) Se coloca un cuadrado de 3x3 LEDs boca abajo (conviene utilizar el mismo apoyo que en el apartado 1), que será el nivel superior (nivel 3) y encima de él, también boca abajo otro cuadrado de 3x3 LEDs, que será el nivel 2. Resulta ventajoso conseguir que las patillas del nivel 3 sujeten las cabezas de los LEDs del nivel 2 antes siquiera de empezar a soldar, porque facilita el proceso.

Tras haber soldado todos los LEDs, una forma sencilla de comprobar si se han cometido errores es cortocircuitar los ánodos de los 2 niveles con un hilo de cobre y verificar con el polímetro si al tocar ánodo y cátodo, se encienden los LEDs correspondientes de los 2 niveles.

- 6) Paso análogo al 5 pero soldando el nivel 1 al conjunto anterior. La comprobación también es semejante pero ahora se deberán encender los LEDs de los 3 niveles.
- 7) Se debe soldar el cubo 3x3x3 al PCB. No convenía doblar las patillas de los LEDs del nivel inferior para que en este paso sea más sencillo que pasen a través de los agujeros del PCB.

Se hace que las 3 patillas de cada LED atraviesen sus respectivos agujeros y se dobla tan sólo una pequeña punta antes de continuar con el siguiente LED (si no se doblan estas puntas, al querer meter el siguiente LED se saldrá el anterior).

Tras introducir todos los LEDs en sus respectivas posiciones se sueldan al PCB. En este momento es aconsejable repetir la comprobación del paso 6, porque probablemente alguna soldadura se haya roto o desoldado.

- 8) Finalmente, soldar 3 cables en un extremo a los terminales correspondientes a los colectores de los 3 transistores y en el otro extremo al ánodo del respectivo nivel.

Por tercera vez resulta conveniente repetir la misma comprobación que en los pasos 6 y 7. Si esta tiene éxito o, en caso contrario, tras corregir los problemas, ya se puede empezar a hacer comprobaciones del funcionamiento del cubo programando el microcontrolador.

Debido a la dificultad que supondría combinarla con el montaje del cubo de LEDs y a la escasez de tiempo, en este proyecto no se ha diseñado una carcasa.

El resultado final obtenido es el siguiente:

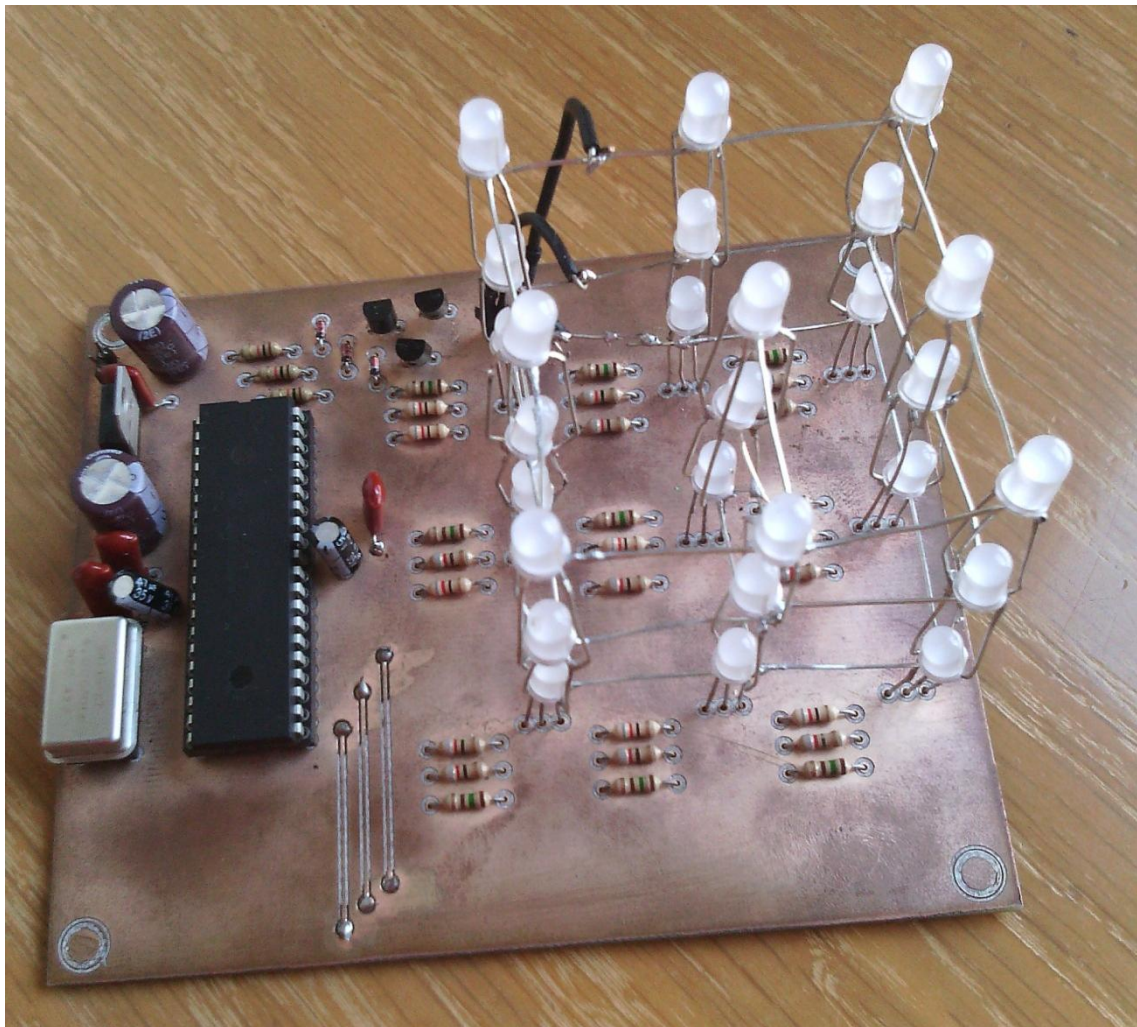


Figura 51. Resultado final cubo de LEDs RGB.

6 Conclusiones

La realización de este trabajo fin de grado ha permitido comprobar que la electrónica es más que meramente programar, actividad con la que se suele asociar esta disciplina. Es necesario efectuar una labor de diseño que implica tener en cuenta cuestiones no sólo electrónicas, sino también prácticas (¿se puede soldar el PCB diseñado sin demasiados problemas?, ¿los periféricos quedan en una posición adecuada para el montaje final?).

Se han utilizado los conocimientos adquiridos a lo largo del grado, tanto de años anteriores (programación en C) como del año actual (diseño de PCBs, diseño 3D) y también se han adquirido otros adicionales, como los vinculados a la edición de vídeos para realizar los vídeos de presentación de los proyectos.

Asimismo, también se han publicado artículos en inglés sobre el Reloj Despertador y el Laser-shooting Robot en la página web DesignSpark, vinculada a la empresa electrónica RS Components. El objetivo perseguido era simplemente dar a conocer los proyectos efectuados. No obstante, se puede considerar que han tenido una cierta difusión, ya que en el momento de la redacción de esta conclusión sumaban más de 150 visitas entre ambos.

Sin embargo, ello no supone que no existan cosas a mejorar. Un aspecto que ha resultado problemático a la larga es no implementar un conector RJ11 para el debugger en los PCBs diseñados. Se esperaba volcar al microcontrolador el programa definitivo en la placa PICDEM 2 PLUS y luego simplemente introducirlo en el zócalo del PCB.

El problema encontrado es que se han tenido que realizar modificaciones varias veces en estos programas. Sacar e introducir continuamente el microcontrolador en el zócalo aumenta el riesgo de que se rompa alguna de sus 40 patillas. Si alguna de estas se quiebra, en función de cuál sea, el microcontrolador se vuelve inútil para el proyecto.

Además, se ha aprendido que para conseguir que un circuito funcione hay que tener paciencia y perseverar, porque resulta habitual resolver un fallo y encontrar otros dos de forma inmediata.

Estos problemas han mejorado la capacidad para detectar errores, partiendo de lo principal (alimentación, oscilador), aquello imprescindible para el funcionamiento del circuito, y siguiendo por lo específico tras asegurar lo anterior. La búsqueda de errores y las pruebas de funcionamiento también han contribuido a ganar agilidad en el manejo de instrumentos como el polímetro, el osciloscopio o el generador de funciones.

En conclusión, se está satisfecho con el trabajo realizado y la experiencia y conocimiento adquiridos en el desarrollo del mismo.

7 Bibliografía

• General

- Dillmann, P.; MacroFab (2017). *Attaching your project to things. Mounting holes and what to watch out for*. Recuperado de: <https://macrofab.com/blog/attaching-your-project-to-things-mounting-holes-and-what-to-watch-out-for/>
- Hausherr, T.; Tom Hausherr's Blog; Mentor Graphics Corporation (2011). *PCB Design Perfection Starts in the CAD Library – Part 13*. Recuperado de: <https://blogs.mentor.com/tom-hausherr/blog/tag/pcb-mounting-holes/>
- Microchip (2012). *Datasheet PIC16F87XA, 28/40/44-Pin Enhanced Flash Microcontrollers*. Recuperado de: <http://ww1.microchip.com/downloads/en/DeviceDoc/39582C.pdf>
- PCB Libraries. *Class Designation Letters For Reference Designations*. Recuperado de: <http://electronica.ugr.es/~amroldan/pcb/2007/modulos/temas/ReferenceDesignators.pdf>
- Rowlett, R; Center for Mathematics and Science Education; University of North Carolina at Chapel Hill, UNC (2005). *How Many? A Dictionary of Units of Measurement*. <http://www.unc.edu/~rowlett/units/>

• Reloj Despertador

- Hitachi (1998). *Datasheet HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver)*. Recuperado de: <https://www.sparkfun.com/datasheets/LCD/HD44780.pdf>
- Maxim Integrated (2005). *Datasheet DS3231, Extremely Accurate I²C-Integrated RTC/TCXO/Crystal*. Recuperado de: <https://datasheets.maximintegrated.com/en/ds/DS3231.pdf>
- Morales Medina, M.A.; Gaussianos (2006). *Cómo calcular qué día de la semana fue*. Recuperado de: <http://gaussianos.com/como-calcular-que-dia-de-la-semana-fue/>
- Prometec. *Displays LCD. Usando displays de texto LCD 16x2 con Arduino*. Recuperado de: <http://www.prometec.net/displays-lcd/>

• Laser-shooting robot

- Hellwig; W, artículo de la revista de manualidades Make:. *Build an Infrared Shooting Arcade*. Recuperado de: <http://makezine.com/projects/make-43/infrared-shooting-arcade/>
- Kostić, D; doctor de la Universidad Técnica de Eindhoven, TU/e (2009). *Introduction Robotics, lecture 1 of 7. Dynamics and Control*. Recuperado de: http://www.es.ele.tue.nl/education/5HC99/wiki/images/7/70/Introduction_Robotics_lecture1.pdf
- Micropik.com. Su tienda de componentes (2017). *SG90 9g Tower Pro*. Recuperado de: <http://www.micropik.com/PDF/SG90Servo.pdf>

Sciavico, L.; Siciliano, B. (2005). *Modelling and Control of Robot Manipulators*. Recuperado de: <https://play.google.com/books/reader?id=hHrkBwAAQBAJ&printsec=frontcover&output=reader&hl=es&pg=GBS.PA9>

Wikipedia. *Ángulos de navegación*. Recuperado de: https://es.wikipedia.org/wiki/%C3%81ngulos_de_navegaci%C3%B3n

- **Cubo de LEDs RGB**

Muy Fácil de Hacer (2015). *Cómo Hacer Cubo De LED 3x3x3 (muy fácil de hacer)*. Vídeo visionado en: <https://www.youtube.com/watch?v=rw2tA0LZgwY>

8 Anexos

Anexo I: Hoja de características módulo RTC DS3231

Anexo II: Partituras de las canciones del Reloj Despertador

DS3231

Extremely Accurate I2C-Integrated RTC/TCXO/Crystal

General Description

The DS3231 is a low-cost, extremely accurate I2C real-time clock (RTC) with an integrated temperature-compensated crystal oscillator (TCXO) and crystal. The device incorporates a battery input, and maintains accurate timekeeping when main power to the device is interrupted. The integration of the crystal resonator enhances the long-term accuracy of the device as well as reduces the piece-part count in a manufacturing line. The DS3231 is available in commercial and industrial temperature ranges, and is offered in a 16-pin, 300-mil SO package.

The RTC maintains seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with an AM/PM indicator. Two programmable time-of-day alarms and a programmable square-wave output are provided. Address and data are transferred serially through an I2C bidirectional bus.

A precision temperature-compensated voltage reference and comparator circuit monitors the status of V_{CC} to detect power failures, to provide a reset output, and to automatically switch to the backup supply when necessary. Additionally, the RST pin is monitored as a pushbutton input for generating a μP reset.

Benefits and Features

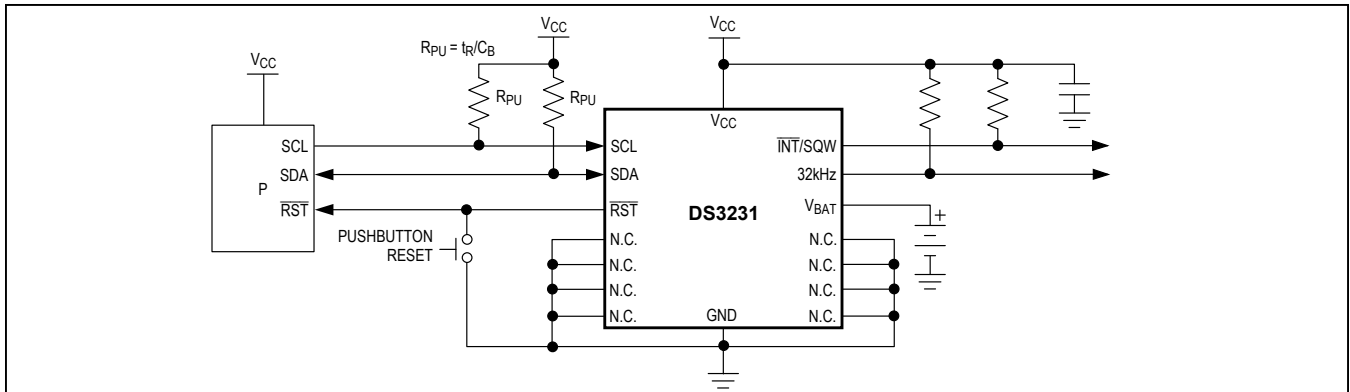
- Highly Accurate RTC Completely Manages All Timekeeping Functions
 - Real-Time Clock Counts Seconds, Minutes, Hours, Date of the Month, Month, Day of the Week, and Year, with Leap-Year Compensation Valid Up to 2100
 - Accuracy $\pm 2\text{ppm}$ from 0°C to $+40^{\circ}\text{C}$
 - Accuracy $\pm 3.5\text{ppm}$ from -40°C to $+85^{\circ}\text{C}$
 - Digital Temp Sensor Output: $\pm 3^{\circ}\text{C}$ Accuracy
 - Register for Aging Trim
 - RST Output/Pushbutton Reset Debounce Input
 - Two Time-of-Day Alarms
 - Programmable Square-Wave Output Signal
- Simple Serial Interface Connects to Most Microcontrollers
 - Fast (400kHz) I2C Interface
- Battery-Backup Input for Continuous Timekeeping
 - Low Power Operation Extends Battery-Backup Run Time
 - 3.3V Operation
- Operating Temperature Ranges: Commercial (0°C to $+70^{\circ}\text{C}$) and Industrial (-40°C to $+85^{\circ}\text{C}$)
- Underwriters Laboratories® (UL) Recognized

Applications

- Servers
- Telematics
- Utility Power Meters
- GPS

Ordering Information and Pin Configuration appear at end of data sheet.

Typical Operating Circuit



Underwriters Laboratories is a registered certification mark of Underwriters Laboratories Inc.

Absolute Maximum Ratings

Voltage Range on Any Pin Relative to Ground-0.3V to +6.0V
 Junction-to-Ambient Thermal Resistance (θ_{JA}) (Note 1) 73°C/W
 Junction-to-Case Thermal Resistance (θ_{JC}) (Note 1)23°C/W
 Operating Temperature Range

DS3231S0°C to +70°C

DS3231SN.....-40°C to +85°C

Junction Temperature+125°C

Storage Temperature Range-40°C to +85°C

Lead Temperature (soldering, 10s)+260°C

Soldering Temperature (reflow, 2 times max)+260°C
 (see the *Handling, PCB Layout, and Assembly* section)

Note 1: Package thermal resistances were obtained using the method described in JEDEC specification JESD51-7, using a four-layer board. For detailed information on package thermal considerations, refer to www.maximintegrated.com/thermal-tutorial.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Recommended Operating Conditions

($T_A = T_{MIN}$ to T_{MAX} , unless otherwise noted.) (Notes 2, 3)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Supply Voltage	V_{CC}		2.3	3.3	5.5	V
	V_{BAT}		2.3	3.0	5.5	V
Logic 1 Input SDA, SCL	V_{IH}		0.7 x V_{CC}		$V_{CC} + 0.3$	V
Logic 0 Input SDA, SCL	V_{IL}		-0.3		0.3 x V_{CC}	V

Electrical Characteristics

($V_{CC} = 2.3V$ to $5.5V$, $V_{CC} =$ Active Supply (see Table 1), $T_A = T_{MIN}$ to T_{MAX} , unless otherwise noted.) (Typical values are at $V_{CC} = 3.3V$, $V_{BAT} = 3.0V$, and $T_A = +25^\circ C$, unless otherwise noted.) (Notes 2, 3)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Active Supply Current	I_{CCA}	(Notes 4, 5)	$V_{CC} = 3.63V$		200	μA
			$V_{CC} = 5.5V$		300	
Standby Supply Current	I_{CCS}	I ² C bus inactive, 32kHz output on, SQW output off (Note 5)	$V_{CC} = 3.63V$		110	μA
			$V_{CC} = 5.5V$		170	
Temperature Conversion Current	$I_{CCSCONV}$	I ² C bus inactive, 32kHz output on, SQW output off	$V_{CC} = 3.63V$		575	μA
			$V_{CC} = 5.5V$		650	
Power-Fail Voltage	V_{PF}		2.45	2.575	2.70	V
Logic 0 Output, 32kHz, \overline{INT}/SQW , SDA	V_{OL}	$I_{OL} = 3mA$			0.4	V
Logic 0 Output, \overline{RST}	V_{OL}	$I_{OL} = 1mA$			0.4	V
Output Leakage Current 32kHz, \overline{INT}/SQW , SDA	I_{LO}	Output high impedance	-1	0	+1	μA
Input Leakage SCL	I_{LI}		-1		+1	μA
\overline{RST} Pin I/O Leakage	I_{OL}	\overline{RST} high impedance (Note 6)	-200		+10	μA
V_{BAT} Leakage Current (V_{CC} Active)	I_{BATLKG}			25	100	nA

Electrical Characteristics (continued)

($V_{CC} = 2.3V$ to $5.5V$, V_{CC} = Active Supply (see Table 1), $T_A = T_{MIN}$ to T_{MAX} , unless otherwise noted.) (Typical values are at $V_{CC} = 3.3V$, $V_{BAT} = 3.0V$, and $T_A = +25^{\circ}C$, unless otherwise noted.) (Notes 2, 3)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Output Frequency	f_{OUT}	$V_{CC} = 3.3V$ or $V_{BAT} = 3.3V$		32.768		kHz
Frequency Stability vs. Temperature (Commercial)	$\Delta f/f_{OUT}$	$V_{CC} = 3.3V$ or $V_{BAT} = 3.3V$, aging offset = 00h	$0^{\circ}C$ to $+40^{\circ}C$		± 2	ppm
			$>40^{\circ}C$ to $+70^{\circ}C$		± 3.5	
Frequency Stability vs. Temperature (Industrial)	$\Delta f/f_{OUT}$	$V_{CC} = 3.3V$ or $V_{BAT} = 3.3V$, aging offset = 00h	$-40^{\circ}C$ to $<0^{\circ}C$		± 3.5	ppm
			$0^{\circ}C$ to $+40^{\circ}C$		± 2	
			$>40^{\circ}C$ to $+85^{\circ}C$		± 3.5	
Frequency Stability vs. Voltage	$\Delta f/V$			1		ppm/V
Trim Register Frequency Sensitivity per LSB	$\Delta f/LSB$	Specified at:	$-40^{\circ}C$	0.7		ppm
			$+25^{\circ}C$	0.1		
			$+70^{\circ}C$	0.4		
			$+85^{\circ}C$	0.8		
Temperature Accuracy	Temp	$V_{CC} = 3.3V$ or $V_{BAT} = 3.3V$	-3		+3	$^{\circ}C$
Crystal Aging	$\Delta f/f_O$	After reflow, not production tested	First year	± 1.0		ppm
			0–10 years	± 5.0		

Electrical Characteristics

($V_{CC} = 0V$, $V_{BAT} = 2.3V$ to $5.5V$, $T_A = T_{MIN}$ to T_{MAX} , unless otherwise noted.) (Note 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Active Battery Current	I_{BATA}	$\overline{EOSC} = 0$, BBSQW = 0, SCL = 400kHz (Note 5)	$V_{BAT} = 3.63V$		70	μA
			$V_{BAT} = 5.5V$		150	
Timekeeping Battery Current	I_{BATT}	$\overline{EOSC} = 0$, BBSQW = 0, EN32kHz = 1, SCL = SDA = 0V or SCL = SDA = V_{BAT} (Note 5)	$V_{BAT} = 3.63V$	0.84	3.0	μA
			$V_{BAT} = 5.5V$	1.0	3.5	
Temperature Conversion Current	I_{BATTC}	$\overline{EOSC} = 0$, BBSQW = 0, SCL = SDA = 0V or SCL = SDA = V_{BAT}	$V_{BAT} = 3.63V$		575	μA
			$V_{BAT} = 5.5V$		650	
Data-Retention Current	I_{BATDR}	$\overline{EOSC} = 1$, SCL = SDA = 0V, $+25^{\circ}C$			100	nA

AC Electrical Characteristics

($V_{CC} = V_{CC(MIN)}$ to $V_{CC(MAX)}$ or $V_{BAT} = V_{BAT(MIN)}$ to $V_{BAT(MAX)}$, $V_{BAT} > V_{CC}$, $T_A = T_{MIN}$ to T_{MAX} , unless otherwise noted.) (Note 2)

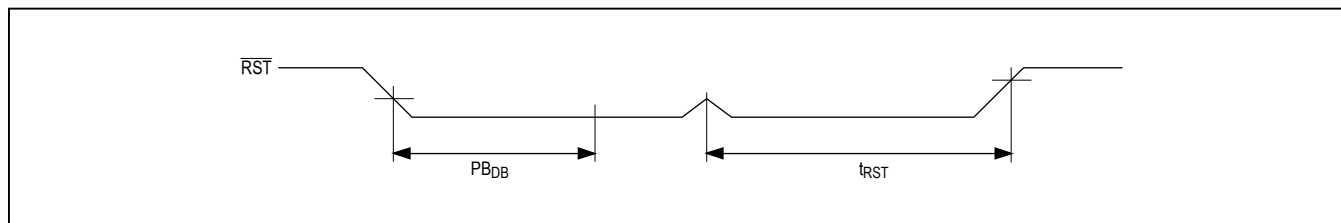
PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
SCL Clock Frequency	f_{SCL}	Fast mode	100		400	kHz
		Standard mode	0		100	
Bus Free Time Between STOP and START Conditions	t_{BUF}	Fast mode	1.3			μs
		Standard mode	4.7			
Hold Time (Repeated) START Condition (Note 7)	$t_{HD:STA}$	Fast mode	0.6			μs
		Standard mode	4.0			
Low Period of SCL Clock	t_{LOW}	Fast mode	1.3			μs
		Standard mode	4.7			
High Period of SCL Clock	t_{HIGH}	Fast mode	0.6			μs
		Standard mode	4.0			
Data Hold Time (Notes 8, 9)	$t_{HD:DAT}$	Fast mode	0		0.9	μs
		Standard mode	0		0.9	
Data Setup Time (Note 10)	$t_{SU:DAT}$	Fast mode	100			ns
		Standard mode	250			
START Setup Time	$t_{SU:STA}$	Fast mode	0.6			μs
		Standard mode	4.7			
Rise Time of Both SDA and SCL Signals (Note 11)	t_R	Fast mode	20 +		300	ns
		Standard mode	0.1 C_B		1000	
Fall Time of Both SDA and SCL Signals (Note 11)	t_F	Fast mode	20 +		300	ns
		Standard mode	0.1 C_B		300	
Setup Time for STOP Condition	$t_{SU:STO}$	Fast mode	0.6			μs
		Standard mode	4.7			
Capacitive Load for Each Bus Line	C_B	(Note 11)			400	pF
Capacitance for SDA, SCL	$C_{I/O}$			10		pF
Pulse Width of Spikes That Must Be Suppressed by the Input Filter	t_{SP}			30		ns
Pushbutton Debounce	PB_{DB}			250		ms
Reset Active Time	t_{RST}			250		ms
Oscillator Stop Flag (OSF) Delay	t_{OSF}	(Note 12)		100		ms
Temperature Conversion Time	t_{CONV}			125	200	ms

Power-Switch Characteristics

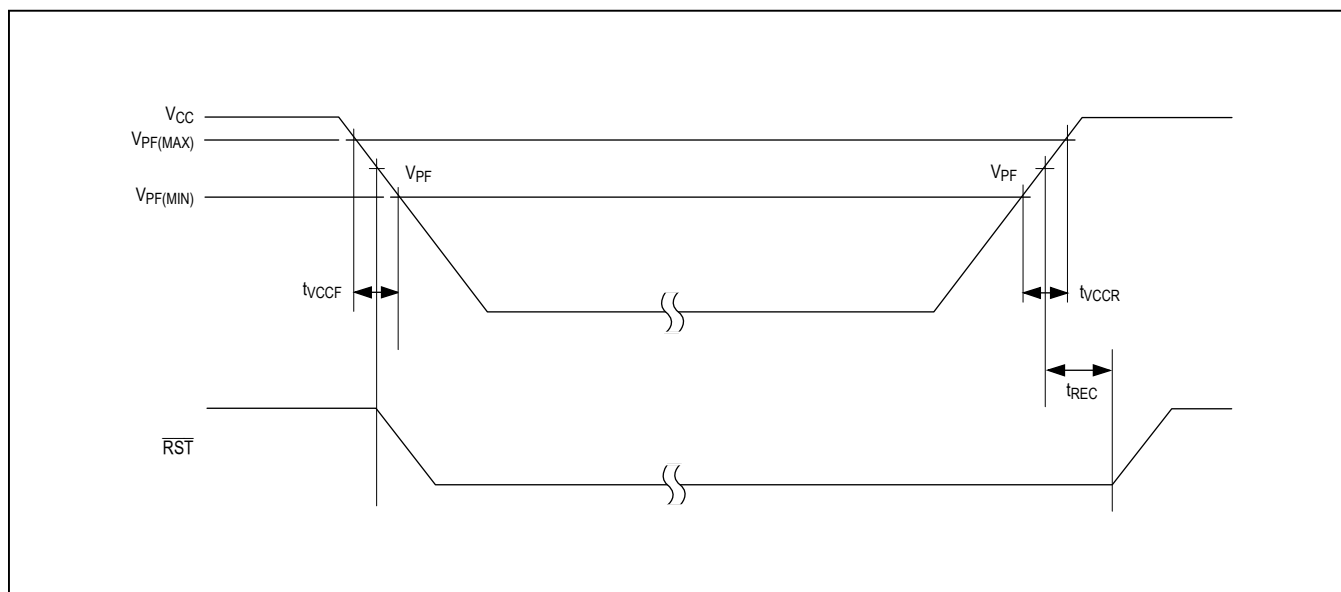
($T_A = T_{MIN}$ to T_{MAX})

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
V_{CC} Fall Time; $V_{PF(MAX)}$ to $V_{PF(MIN)}$	t_{VCCF}		300			μs
V_{CC} Rise Time; $V_{PF(MIN)}$ to $V_{PF(MAX)}$	t_{VCCR}		0			μs
Recovery at Power-Up	t_{REC}	(Note 13)		250	300	ms

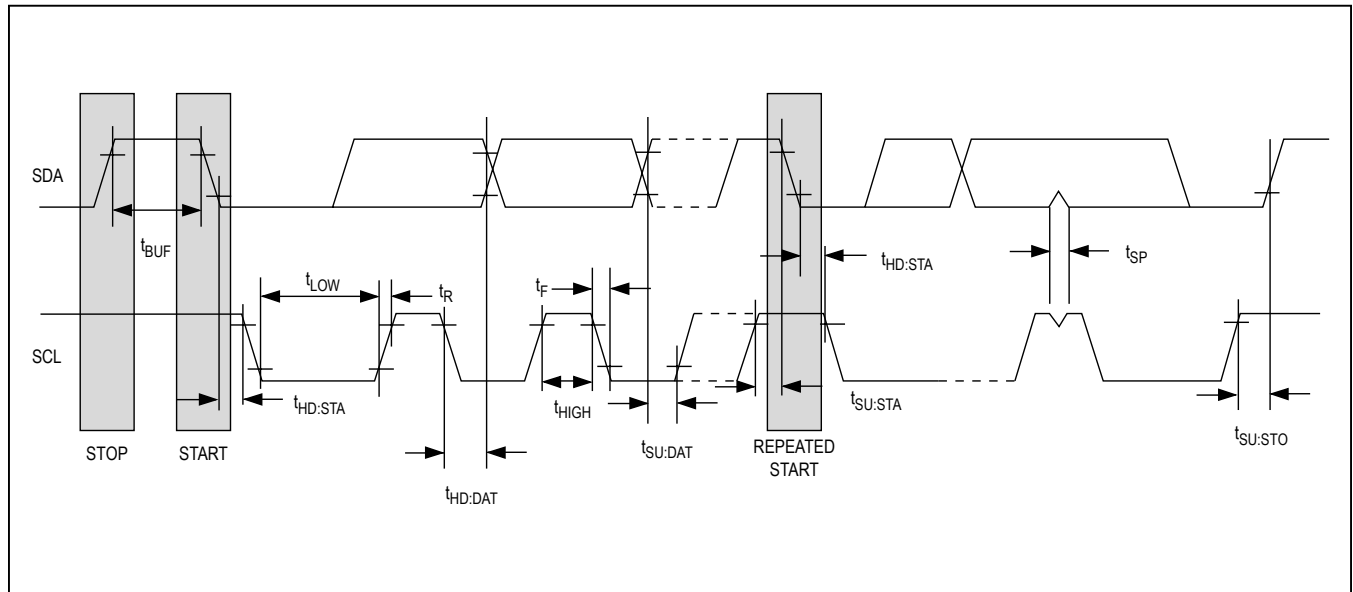
Pushbutton Reset Timing



Power-Switch Timing



Data Transfer on I2C Serial Bus



WARNING: Negative undershoots below -0.3V while the part is in battery-backed mode may cause loss of data.

Note 2: Limits at -40°C are guaranteed by design and not production tested.

Note 3: All voltages are referenced to ground.

Note 4: I_{CCA} —SCL clocking at max frequency = 400kHz.

Note 5: Current is the averaged input current, which includes the temperature conversion current.

Note 6: The \overline{RST} pin has an internal 50k Ω (nominal) pullup resistor to V_{CC} .

Note 7: After this period, the first clock pulse is generated.

Note 8: A device must internally provide a hold time of at least 300ns for the SDA signal (referred to the $V_{IH(MIN)}$ of the SCL signal) to bridge the undefined region of the falling edge of SCL.

Note 9: The maximum $t_{HD:DAT}$ needs only to be met if the device does not stretch the low period (t_{LOW}) of the SCL signal.

Note 10: A fast-mode device can be used in a standard-mode system, but the requirement $t_{SU:DAT} \geq 250\text{ns}$ must then be met. This is automatically the case if the device does not stretch the low period of the SCL signal. If such a device does stretch the low period of the SCL signal, it must output the next data bit to the SDA line $t_{R(MAX)} + t_{SU:DAT} = 1000 + 250 = 1250\text{ns}$ before the SCL line is released.

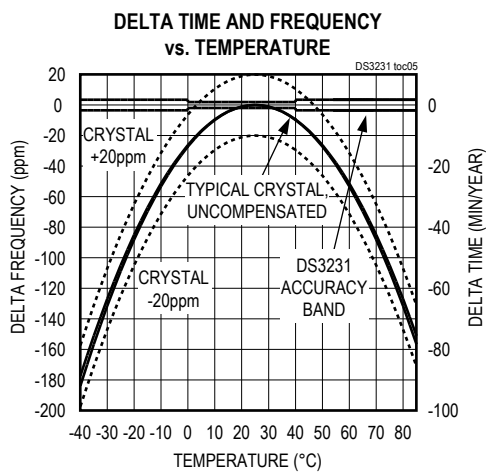
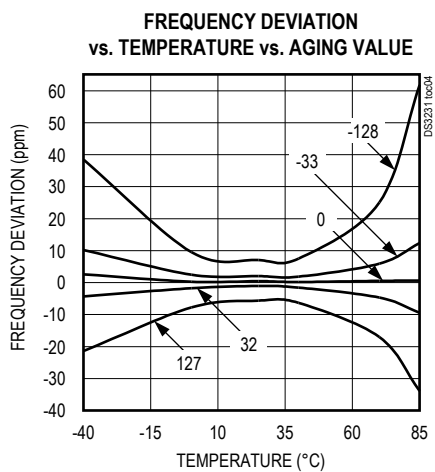
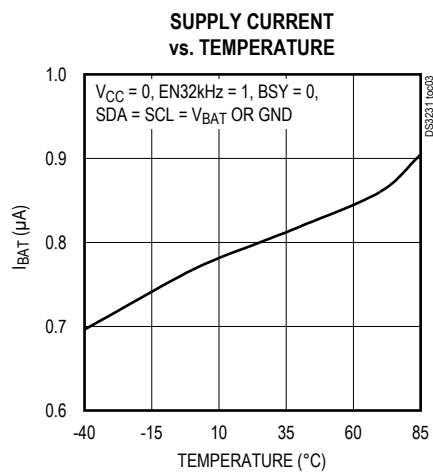
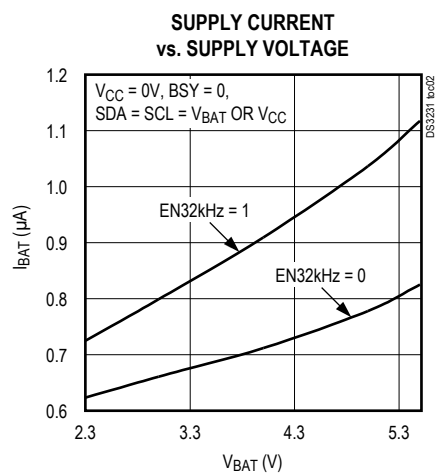
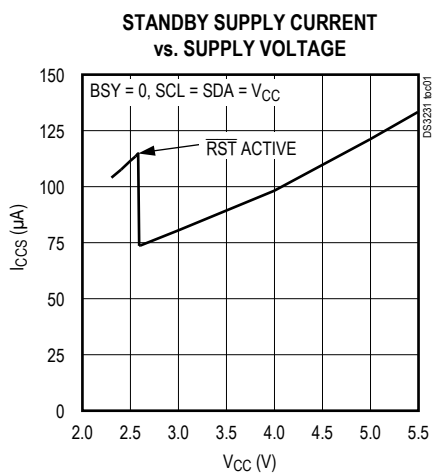
Note 11: C_B —total capacitance of one bus line in pF.

Note 12: The parameter t_{OSF} is the period of time the oscillator must be stopped for the OSF flag to be set over the voltage range of $0.0\text{V} \leq V_{CC} \leq V_{CC(MAX)}$ and $2.3\text{V} \leq V_{BAT} \leq 3.4\text{V}$.

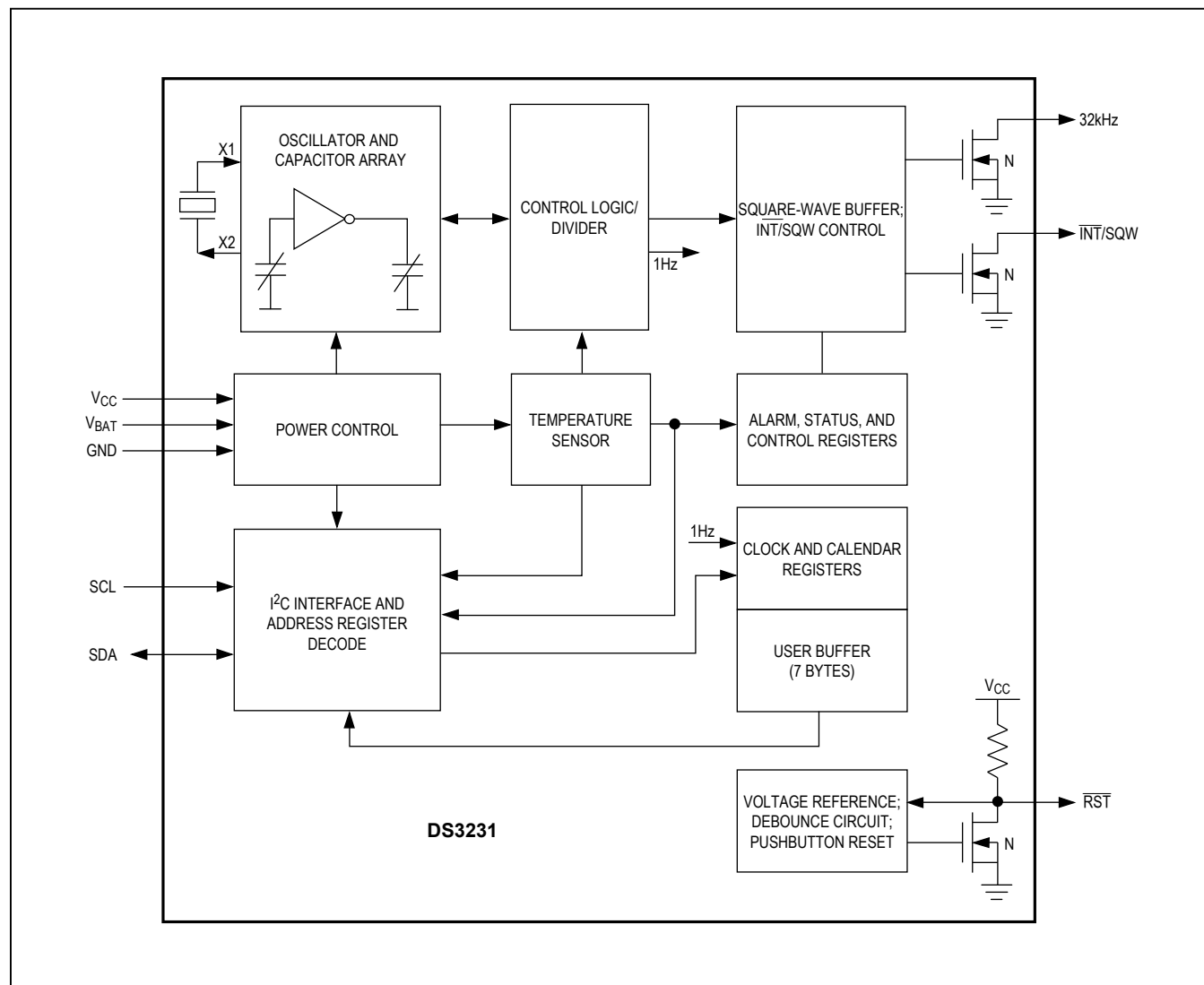
Note 13: This delay applies only if the oscillator is enabled and running. If the \overline{EOSC} bit is a 1, t_{REC} is bypassed and \overline{RST} immediately goes high. The state of \overline{RST} does not affect the I2C interface, RTC, or TCXO.

Typical Operating Characteristics

($V_{CC} = +3.3V$, $T_A = +25^\circ C$, unless otherwise noted.)



Block Diagram



Pin Description

PIN	NAME	FUNCTION
1	32kHz	32kHz Output. This open-drain pin requires an external pullup resistor. When enabled, the output operates on either power supply. It may be left open if not used.
2	V _{CC}	DC Power Pin for Primary Power Supply. This pin should be decoupled using a 0.1μF to 1.0μF capacitor. If not used, connect to ground.
3	INT/SQW	Active-Low Interrupt or Square-Wave Output. This open-drain pin requires an external pullup resistor connected to a supply at 5.5V or less. This multifunction pin is determined by the state of the INTCN bit in the Control Register (0Eh). When INTCN is set to logic 0, this pin outputs a square wave and its frequency is determined by RS2 and RS1 bits. When INTCN is set to logic 1, then a match between the timekeeping registers and either of the alarm registers activates the INT/SQW pin (if the alarm is enabled). Because the INTCN bit is set to logic 1 when power is first applied, the pin defaults to an interrupt output with alarms disabled. The pullup voltage can be up to 5.5V, regardless of the voltage on V _{CC} . If not used, this pin can be left unconnected.
4	RST	Active-Low Reset. This pin is an open-drain input/output. It indicates the status of V _{CC} relative to the V _{PF} specification. As V _{CC} falls below V _{PF} , the RST pin is driven low. When V _{CC} exceeds V _{PF} , for t _{RST} , the RST pin is pulled high by the internal pullup resistor. The active-low, open-drain output is combined with a debounced pushbutton input function. This pin can be activated by a pushbutton reset request. It has an internal 50kΩ nominal value pullup resistor to V _{CC} . No external pullup resistors should be connected. If the oscillator is disabled, t _{REC} is bypassed and RST immediately goes high.
5–12	N.C.	No Connection. Must be connected to ground.
13	GND	Ground
14	V _{BAT}	Backup Power-Supply Input. When using the device with the V _{BAT} input as the primary power source, this pin should be decoupled using a 0.1μF to 1.0μF low-leakage capacitor. When using the device with the V _{BAT} input as the backup power source, the capacitor is not required. If V _{BAT} is not used, connect to ground. The device is UL recognized to ensure against reverse charging when used with a primary lithium battery. Go to www.maximintegrated.com/qa/info/ul .
15	SDA	Serial Data Input/Output. This pin is the data input/output for the I ² C serial interface. This open-drain pin requires an external pullup resistor. The pullup voltage can be up to 5.5V, regardless of the voltage on V _{CC} .
16	SCL	Serial Clock Input. This pin is the clock input for the I ² C serial interface and is used to synchronize data movement on the serial interface. Up to 5.5V can be used for this pin, regardless of the voltage on V _{CC} .

Detailed Description

The DS3231 is a serial RTC driven by a temperature-compensated 32kHz crystal oscillator. The TCXO provides a stable and accurate reference clock, and maintains the RTC to within ±2 minutes per year accuracy from -40°C to +85°C. The TCXO frequency output is available at the 32kHz pin. The RTC is a low-power clock/calendar with two programmable time-of-day alarms and a programmable square-wave output. The INT/SQW provides either an interrupt signal due to alarm conditions or a square-wave output. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap

year. The clock operates in either the 24-hour or 12-hour format with an AM/PM indicator. The internal registers are accessible through an I²C bus interface.

A temperature-compensated voltage reference and comparator circuit monitors the level of V_{CC} to detect power failures and to automatically switch to the backup supply when necessary. The RST pin provides an external pushbutton function and acts as an indicator of a power-fail event.

Operation

The block diagram shows the main elements of the DS3231. The eight blocks can be grouped into four functional groups: TCXO, power control, pushbutton function, and RTC. Their operations are described separately in the following sections.

32kHz TCXO

The temperature sensor, oscillator, and control logic form the TCXO. The controller reads the output of the on-chip temperature sensor and uses a lookup table to determine the capacitance required, adds the aging correction in AGE register, and then sets the capacitance selection registers. New values, including changes to the AGE register, are loaded only when a change in the temperature value occurs, or when a user-initiated temperature conversion is completed. Temperature conversion occurs on initial application of V_{CC} and once every 64 seconds afterwards.

Power Control

This function is provided by a temperature-compensated voltage reference and a comparator circuit that monitors the V_{CC} level. When V_{CC} is greater than V_{PF} , the part is powered by V_{CC} . When V_{CC} is less than V_{PF} but greater than V_{BAT} , the DS3231 is powered by V_{CC} . If V_{CC} is less than V_{PF} and is less than V_{BAT} , the device is powered by V_{BAT} . See Table 1.

Table 1. Power Control

SUPPLY CONDITION	ACTIVE SUPPLY
$V_{CC} < V_{PF}$, $V_{CC} < V_{BAT}$	V_{BAT}
$V_{CC} < V_{PF}$, $V_{CC} > V_{BAT}$	V_{CC}
$V_{CC} > V_{PF}$, $V_{CC} < V_{BAT}$	V_{CC}
$V_{CC} > V_{PF}$, $V_{CC} > V_{BAT}$	V_{CC}

To preserve the battery, the first time V_{BAT} is applied to the device, the oscillator will not start up until V_{CC} exceeds V_{PF} , or until a valid I²C address is written to the part. Typical oscillator startup time is less than one second. Approximately 2 seconds after V_{CC} is applied, or a valid I²C address is written, the device makes a temperature measurement and applies the calculated correction to the oscillator. Once the oscillator is running, it continues to run as long as a valid power source is available (V_{CC} or V_{BAT}), and the device continues to measure the temperature and correct the oscillator frequency every 64 seconds.

On the first application of power (V_{CC}) or when a valid I²C address is written to the part (V_{BAT}), the time and date registers are reset to 01/01/00 01 00:00:00 (DD/MM/YY DOW HH:MM:SS).

V_{BAT} Operation

There are several modes of operation that affect the amount of V_{BAT} current that is drawn. While the device

is powered by V_{BAT} and the serial interface is active, active battery current, I_{BATA} , is drawn. When the serial interface is inactive, timekeeping current (I_{BATT}), which includes the averaged temperature conversion current, I_{BATTC} , is used (refer to Application Note 3644: *Power Considerations for Accurate Real-Time Clocks* for details). Temperature conversion current, I_{BATTC} , is specified since the system must be able to support the periodic higher current pulse and still maintain a valid voltage level. Data retention current, I_{BATTD} , is the current drawn by the part when the oscillator is stopped ($\overline{EOSC} = 1$). This mode can be used to minimize battery requirements for times when maintaining time and date information is not necessary, e.g., while the end system is waiting to be shipped to a customer.

Pushbutton Reset Function

The DS3231 provides for a pushbutton switch to be connected to the \overline{RST} output pin. When the DS3231 is not in a reset cycle, it continuously monitors the \overline{RST} signal for a low going edge. If an edge transition is detected, the DS3231 debounces the switch by pulling the \overline{RST} low. After the internal timer has expired (PB_{DB}), the DS3231 continues to monitor the \overline{RST} line. If the line is still low, the DS3231 continuously monitors the line looking for a rising edge. Upon detecting release, the DS3231 forces the \overline{RST} pin low and holds it low for t_{RST} .

\overline{RST} is also used to indicate a power-fail condition. When V_{CC} is lower than V_{PF} , an internal power-fail signal is generated, which forces the \overline{RST} pin low. When V_{CC} returns to a level above V_{PF} , the \overline{RST} pin is held low for approximately 250ms (t_{REC}) to allow the power supply to stabilize. If the oscillator is not running (see the *Power Control* section) when V_{CC} is applied, t_{REC} is bypassed and \overline{RST} immediately goes high. Assertion of the \overline{RST} output, whether by pushbutton or power-fail detection, does not affect the internal operation of the DS3231.

Real-Time Clock

With the clock source from the TCXO, the RTC provides seconds, minutes, hours, day, date, month, and year information. The date at the end of the month is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with an AM/PM indicator.

The clock provides two programmable time-of-day alarms and a programmable square-wave output. The INT/SQW pin either generates an interrupt due to alarm condition or outputs a square-wave signal and the selection is controlled by the bit $INTCN$.

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/24	AM/PM 20 Hour	10 Hour	Hour				Hours	1–12 + AM/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date			Date			Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/24	AM/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + AM/PM 00–23
0Ah	A1M4	DY/DT	10 Date			Day			Alarm 1 Day	1–7
						Date			Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/24	AM/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + AM/PM 00–23
0Dh	A2M4	DY/DT	10 Date			Day			Alarm 2 Day	1–7
						Date			Alarm 2 Date	1–31
0Eh	EOSC	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—

Figure 1. Timekeeping Registers

Note: Unless otherwise specified, the registers' state is not defined when power is first applied.

Address Map

Figure 1 shows the address map for the DS3231 timekeeping registers. During a multibyte access, when the address pointer reaches the end of the register space (12h), it wraps around to location 00h. On an I2C START or address pointer incrementing to location 00h, the current time is transferred to a second set of registers. The time information is read from these secondary registers, while the clock may continue to run. This eliminates the need to reread the registers in case the main registers update during a read.

I2C Interface

The I2C interface is accessible whenever either V_{CC} or V_{BAT} is at a valid level. If a microcontroller connected

to the DS3231 resets because of a loss of V_{CC} or other event, it is possible that the microcontroller and DS3231 I2C communications could become unsynchronized, e.g., the microcontroller resets while reading data from the DS3231. When the microcontroller resets, the DS3231 I2C interface may be placed into a known state by toggling SCL until SDA is observed to be at a high level. At that point the microcontroller should pull SDA low while SCL is high, generating a START condition.

Clock and Calendar

The time and calendar information is obtained by reading the appropriate register bytes. Figure 1 illustrates the RTC registers. The time and calendar data are set or initialized by writing the appropriate register bytes. The contents of the time and calendar registers are in the binary-coded

decimal (BCD) format. The DS3231 can be run in either 12-hour or 24-hour mode. Bit 6 of the hours register is defined as the 12- or 24-hour mode select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic-high being PM. In the 24-hour mode, bit 5 is the 20-hour bit (20–23 hours). The century bit (bit 7 of the month register) is toggled when the years register overflows from 99 to 00.

The day-of-week register increments at midnight. Values that correspond to the day of week are user-defined but must be sequential (i.e., if 1 equals Sunday, then 2 equals Monday, and so on). Illogical time and date entries result in undefined operation.

When reading or writing the time and date registers, secondary (user) buffers are used to prevent errors when the internal registers update. When reading the time and date registers, the user buffers are synchronized to the internal registers on any START and when the register pointer rolls over to zero. The time information is read from these secondary registers, while the clock continues to run. This eliminates the need to reread the registers in case the main registers update during a read.

The countdown chain is reset whenever the seconds register is written. Write transfers occur on the acknowledge from the DS3231. Once the countdown chain is reset, to avoid rollover issues the remaining time and date registers must be written within 1 second. The 1Hz square-wave output, if enabled, transitions high 500ms after the seconds data transfer, provided the oscillator is already running.

Alarms

The DS3231 contains two time-of-day/date alarms. Alarm 1 can be set by writing to registers 07h to 0Ah. Alarm 2 can be set by writing to registers 0Bh to 0Dh. The alarms can be programmed (by the alarm enable and INTCN bits of the control register) to activate the $\overline{\text{INT}}/\text{SQW}$ output on an alarm match condition. Bit 7 of each of the time-of-day/date alarm registers are mask bits (Table 2). When all the mask bits for each alarm are logic 0, an alarm only occurs when the values in the timekeeping registers match the corresponding values stored in the time-of-day/date alarm registers. The alarms can also be programmed to repeat every second, minute, hour, day, or date. Table 2 shows the possible settings. Configurations not listed in the table will result in illogical operation.

The $\text{DY}/\overline{\text{DT}}$ bits (bit 6 of the alarm day/date registers) control whether the alarm value stored in bits 0 to 5 of that register reflects the day of the week or the date of the month. If $\text{DY}/\overline{\text{DT}}$ is written to logic 0, the alarm will be the result of a match with date of the month. If $\text{DY}/\overline{\text{DT}}$ is written to logic 1, the alarm will be the result of a match with day of the week.

When the RTC register values match alarm register settings, the corresponding Alarm Flag 'A1F' or 'A2F' bit is set to logic 1. If the corresponding Alarm Interrupt Enable 'A1IE' or 'A2IE' is also set to logic 1 and the INTCN bit is set to logic 1, the alarm condition will activate the $\overline{\text{INT}}/\text{SQW}$ signal. The match is tested on the once-per-second update of the time and date registers.

Table 2. Alarm Mask Bits

DY/ $\overline{\text{DT}}$	ALARM 1 REGISTER MASK BITS (BIT 7)				ALARM RATE
	A1M4	A1M3	A1M2	A1M1	
X	1	1	1	1	Alarm once per second
X	1	1	1	0	Alarm when seconds match
X	1	1	0	0	Alarm when minutes and seconds match
X	1	0	0	0	Alarm when hours, minutes, and seconds match
0	0	0	0	0	Alarm when date, hours, minutes, and seconds match
1	0	0	0	0	Alarm when day, hours, minutes, and seconds match
DY/ $\overline{\text{DT}}$	ALARM 2 REGISTER MASK BITS (BIT 7)			ALARM RATE	
	A2M4	A2M3	A2M2		
X	1	1	1	Alarm once per minute (00 seconds of every minute)	
X	1	1	0	Alarm when minutes match	
X	1	0	0	Alarm when hours and minutes match	
0	0	0	0	Alarm when date, hours, and minutes match	
1	0	0	0	Alarm when day, hours, and minutes match	

Control Register (0Eh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	$\overline{\text{EOSC}}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE
POR:	0	0	0	1	1	1	0	0

Special-Purpose Registers

The DS3231 has two additional registers (control and status) that control the real-time clock, alarms, and square-wave output.

Control Register (0Eh)

Bit 7: Enable Oscillator ($\overline{\text{EOSC}}$). When set to logic 0, the oscillator is started. When set to logic 1, the oscillator is stopped when the DS3231 switches to V_{BAT} . This bit is clear (logic 0) when power is first applied. When the DS3231 is powered by V_{CC} , the oscillator is always on regardless of the status of the $\overline{\text{EOSC}}$ bit. When $\overline{\text{EOSC}}$ is disabled, all register data is static.

Bit 6: Battery-Backed Square-Wave Enable (BBSQW). When set to logic 1 with $\text{INTCN} = 0$ and $V_{\text{CC}} < V_{\text{PF}}$, this bit enables the square wave. When BBSQW is logic 0, the $\overline{\text{INT}}/\text{SQW}$ pin goes high impedance when $V_{\text{CC}} < V_{\text{PF}}$. This bit is disabled (logic 0) when power is first applied.

Bit 5: Convert Temperature (CONV). Setting this bit to 1 forces the temperature sensor to convert the temperature into digital code and execute the TCXO algorithm to update the capacitance array to the oscillator. This can only happen when a conversion is not already in progress. The user should check the status bit BSY before forcing the controller to start a new TCXO execution. A user-initiated temperature conversion does not affect the internal 64-second update cycle.

A user-initiated temperature conversion does not affect the BSY bit for approximately 2ms. The CONV bit remains at a 1 from the time it is written until the conversion is finished, at which time both CONV and BSY go to 0. The CONV bit should be used when monitoring the status of a user-initiated conversion.

Bits 4 and 3: Rate Select (RS2 and RS1). These bits control the frequency of the square-wave output when

the square wave has been enabled. The following table shows the square-wave frequencies that can be selected with the RS bits. These bits are both set to logic 1 (8.192kHz) when power is first applied.

SQUARE-WAVE OUTPUT FREQUENCY

RS2	RS1	SQUARE-WAVE OUTPUT FREQUENCY
0	0	1Hz
0	1	1.024kHz
1	0	4.096kHz
1	1	8.192kHz

Bit 2: Interrupt Control (INTCN). This bit controls the $\overline{\text{INT}}/\text{SQW}$ signal. When the INTCN bit is set to logic 0, a square wave is output on the $\overline{\text{INT}}/\text{SQW}$ pin. When the INTCN bit is set to logic 1, then a match between the time-keeping registers and either of the alarm registers activates the $\overline{\text{INT}}/\text{SQW}$ output (if the alarm is also enabled). The corresponding alarm flag is always set regardless of the state of the INTCN bit. The INTCN bit is set to logic 1 when power is first applied.

Bit 1: Alarm 2 Interrupt Enable (A2IE). When set to logic 1, this bit permits the alarm 2 flag (A2F) bit in the status register to assert $\overline{\text{INT}}/\text{SQW}$ (when $\text{INTCN} = 1$). When the A2IE bit is set to logic 0 or INTCN is set to logic 0, the A2F bit does not initiate an interrupt signal. The A2IE bit is disabled (logic 0) when power is first applied.

Bit 0: Alarm 1 Interrupt Enable (A1IE). When set to logic 1, this bit permits the alarm 1 flag (A1F) bit in the status register to assert $\overline{\text{INT}}/\text{SQW}$ (when $\text{INTCN} = 1$). When the A1IE bit is set to logic 0 or INTCN is set to logic 0, the A1F bit does not initiate the $\overline{\text{INT}}/\text{SQW}$ signal. The A1IE bit is disabled (logic 0) when power is first applied.

Status Register (0Fh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	OSF	0	0	0	EN32kHz	BSY	A2F	A1F
POR:	1	0	0	0	1	X	X	X

Status Register (0Fh)

Bit 7: Oscillator Stop Flag (OSF). A logic 1 in this bit indicates that the oscillator either is stopped or was stopped for some period and may be used to judge the validity of the timekeeping data. This bit is set to logic 1 any time that the oscillator stops. The following are examples of conditions that can cause the OSF bit to be set:

- 1) The first time power is applied.
- 2) The voltages present on both V_{CC} and V_{BAT} are insufficient to support oscillation.
- 3) The $\overline{E}OSC$ bit is turned off in battery-backed mode.
- 4) External influences on the crystal (i.e., noise, leakage, etc.).

This bit remains at logic 1 until written to logic 0.

Bit 3: Enable 32kHz Output (EN32kHz). This bit controls the status of the 32kHz pin. When set to logic 1, the 32kHz pin is enabled and outputs a 32.768kHz square-wave signal. When set to logic 0, the 32kHz pin goes to a high-impedance state. The initial power-up state of this bit is logic 1, and a 32.768kHz square-wave signal appears at the 32kHz pin after a power source is applied to the DS3231 (if the oscillator is running).

Bit 2: Busy (BSY). This bit indicates the device is busy executing TCXO functions. It goes to logic 1 when the conversion signal to the temperature sensor is asserted and then is cleared when the device is in the 1-minute idle state.

Bit 1: Alarm 2 Flag (A2F). A logic 1 in the alarm 2 flag bit indicates that the time matched the alarm 2 registers. If the A2IE bit is logic 1 and the INTCN bit is set to logic 1, the \overline{INT}/SQW pin is also asserted. A2F is cleared when written to logic 0. This bit can only be written to logic 0. Attempting to write to logic 1 leaves the value unchanged.

Bit 0: Alarm 1 Flag (A1F). A logic 1 in the alarm 1 flag bit indicates that the time matched the alarm 1 registers. If the

A1IE bit is logic 1 and the INTCN bit is set to logic 1, the \overline{INT}/SQW pin is also asserted. A1F is cleared when written to logic 0. This bit can only be written to logic 0. Attempting to write to logic 1 leaves the value unchanged.

Aging Offset

The aging offset register takes a user-provided value to add to or subtract from the codes in the capacitance array registers. The code is encoded in two's complement, with bit 7 representing the sign bit. One LSB represents one small capacitor to be switched in or out of the capacitance array at the crystal pins. The aging offset register capacitance value is added or subtracted from the capacitance value that the device calculates for each temperature compensation. The offset register is added to the capacitance array during a normal temperature conversion, if the temperature changes from the previous conversion, or during a manual user conversion (setting the CONV bit). To see the effects of the aging register on the 32kHz output frequency immediately, a manual conversion should be started after each aging register change.

Positive aging values add capacitance to the array, slowing the oscillator frequency. Negative values remove capacitance from the array, increasing the oscillator frequency.

The change in ppm per LSB is different at different temperatures. The frequency vs. temperature curve is shifted by the values used in this register. At +25°C, one LSB typically provides about 0.1ppm change in frequency.

Use of the aging register is not needed to achieve the accuracy as defined in the EC tables, but could be used to help compensate for aging at a given temperature. See the *Typical Operating Characteristics* section for a graph showing the effect of the register on accuracy over temperature.

Aging Offset (10h)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	Sign	Data	Data	Data	Data	Data	Data	Data
POR:	0	0	0	0	0	0	0	0

Temperature Register (Upper Byte) (11h)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	Sign	Data	Data	Data	Data	Data	Data	Data
POR:	0	0	0	0	0	0	0	0

Temperature Register (Lower Byte) (12h)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	Data	Data	0	0	0	0	0	0
POR:	0	0	0	0	0	0	0	0

Temperature Registers (11h–12h)

Temperature is represented as a 10-bit code with a resolution of 0.25°C and is accessible at location 11h and 12h. The temperature is encoded in two's complement format. The upper 8 bits, the integer portion, are at location 11h and the lower 2 bits, the fractional portion, are in the upper nibble at location 12h. For example, 00011001 01b = +25.25°C. Upon power reset, the registers are set to a default temperature of 0°C and the controller starts a temperature conversion. The temperature is read on initial application of V_{CC} or I2C access on V_{BAT} and once every 64 seconds afterwards. The temperature registers are updated after each user-initiated conversion and on every 64-second conversion. The temperature registers are read-only.

I2C Serial Data Bus

The DS3231 supports a bidirectional I2C bus and data transmission protocol. A device that sends data onto the bus is defined as a transmitter and a device receiving data is defined as a receiver. The device that controls the message is called a master. The devices that are controlled by the master are slaves. The bus must be controlled by a master device that generates the serial clock (SCL), controls the bus access, and generates the START and STOP conditions. The DS3231 operates as a slave on the I2C bus. Connections to the bus are made through the SCL input and open-drain SDA I/O lines. Within the bus specifications, a standard mode (100kHz maximum clock rate) and a fast mode (400kHz maximum clock rate) are defined. The DS3231 works in both modes.

The following bus protocol has been defined (Figure 2):

- Data transfer may be initiated only when the bus is not busy.
- During data transfer, the data line must remain stable whenever the clock line is high. Changes in the data

line while the clock line is high are interpreted as control signals.

Accordingly, the following bus conditions have been defined:

Bus not busy: Both data and clock lines remain high.

START data transfer: A change in the state of the data line from high to low, while the clock line is high, defines a START condition.

STOP data transfer: A change in the state of the data line from low to high, while the clock line is high, defines a STOP condition.

Data valid: The state of the data line represents valid data when, after a START condition, the data line is stable for the duration of the high period of the clock signal. The data on the line must be changed during the low period of the clock signal. There is one clock pulse per bit of data.

Each data transfer is initiated with a START condition and terminated with a STOP condition. The number of data bytes transferred between the START and the STOP conditions is not limited, and is determined by the master device. The information is transferred byte-wise and each receiver acknowledges with a ninth bit.

Acknowledge: Each receiving device, when addressed, is obliged to generate an acknowledge after the reception of each byte. The master device must generate an extra clock pulse, which is associated with this acknowledge bit.

A device that acknowledges must pull down the SDA line during the acknowledge clock pulse in such a way that the SDA line is stable low during the high period of the acknowledge-related clock pulse. Of course, setup and hold times must be taken into account. A master must signal an end of data to the slave by not generat-

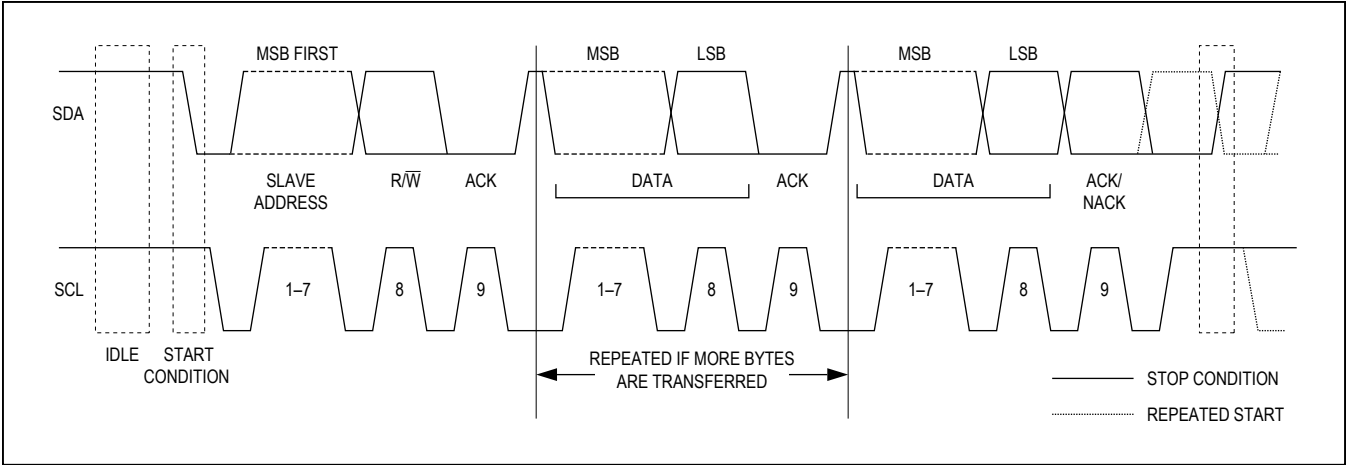


Figure 2. I²C Data Transfer Overview

ing an acknowledge bit on the last byte that has been clocked out of the slave. In this case, the slave must leave the data line high to enable the master to generate the STOP condition.

Figures 3 and 4 detail how data transfer is accomplished on the I²C bus. Depending upon the state of the R/W bit, two types of data transfer are possible:

Data transfer from a master transmitter to a slave receiver. The first byte transmitted by the master

is the slave address. Next follows a number of data bytes. The slave returns an acknowledge bit after each received byte. Data is transferred with the most significant bit (MSB) first.

Data transfer from a slave transmitter to a master receiver. The first byte (the slave address) is transmitted by the master. The slave then returns an acknowledge bit. Next follows a number of data bytes transmitted by the slave to the master. The master returns an acknowledge bit after all received bytes other than the

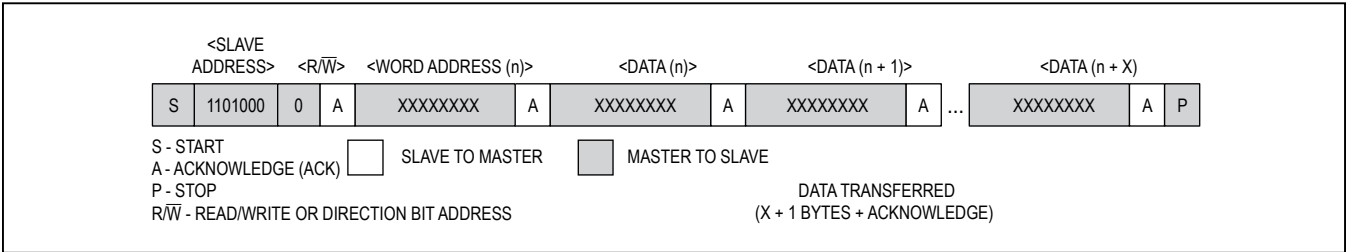


Figure 3. Data Write—Slave Receiver Mode

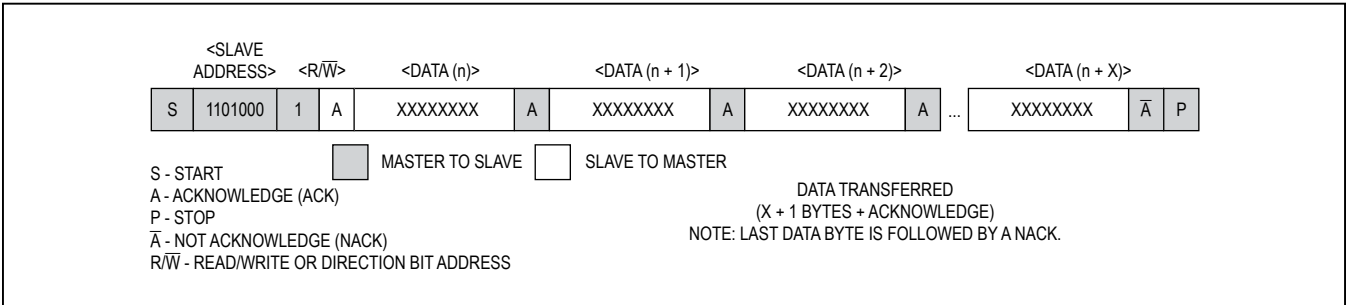


Figure 4. Data Read—Slave Transmitter Mode

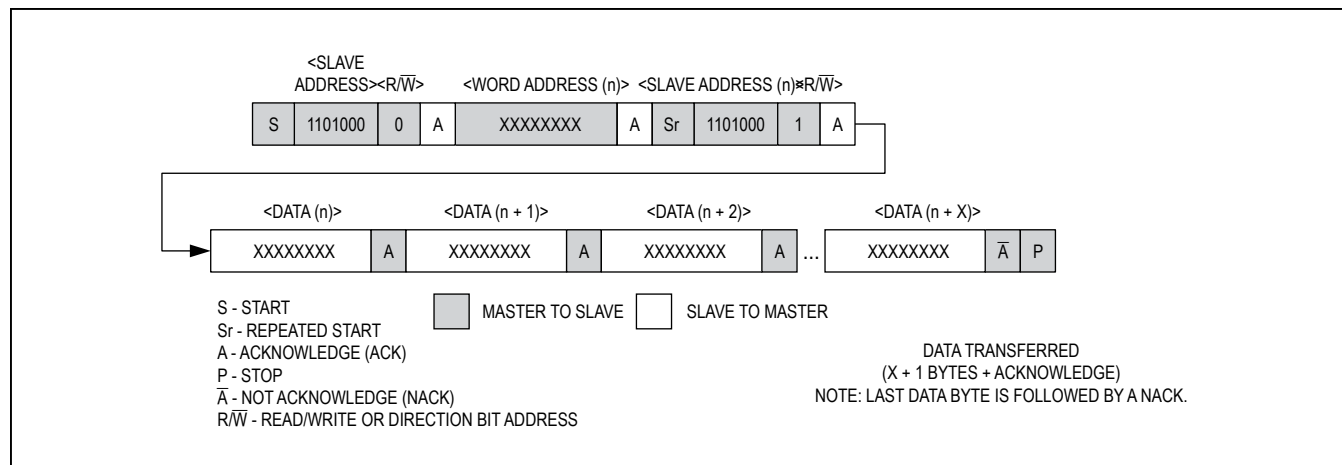


Figure 5. Data Write/Read (Write Pointer, Then Read)—Slave Receive and Transmit

last byte. At the end of the last received byte, a not acknowledge is returned.

The master device generates all the serial clock pulses and the START and STOP conditions. A transfer is ended with a STOP condition or with a repeated START condition. Since a repeated START condition is also the beginning of the next serial transfer, the bus will not be released. Data is transferred with the most significant bit (MSB) first.

The DS3231 can operate in the following two modes:

Slave receiver mode (DS3231 write mode): Serial data and clock are received through SDA and SCL. After each byte is received, an acknowledge bit is transmitted. START and STOP conditions are recognized as the beginning and end of a serial transfer. Address recognition is performed by hardware after reception of the slave address and direction bit. The slave address byte is the first byte received after the master generates the START condition. The slave address byte contains the 7-bit DS3231 address, which is 1101000, followed by the direction bit (R/W), which is 0 for a write. After receiving and decoding the slave address byte, the DS3231 outputs an acknowledge on SDA. After the DS3231 acknowledges the slave address + write bit, the master transmits a word address to the DS3231. This sets the register pointer on the DS3231, with the DS3231 acknowledging the

transfer. The master may then transmit zero or more bytes of data, with the DS3231 acknowledging each byte received. The register pointer increments after each data byte is transferred. The master generates a STOP condition to terminate the data write.

Slave transmitter mode (DS3231 read mode): The first byte is received and handled as in the slave receiver mode. However, in this mode, the direction bit indicates that the transfer direction is reversed. Serial data is transmitted on SDA by the DS3231 while the serial clock is input on SCL. START and STOP conditions are recognized as the beginning and end of a serial transfer. Address recognition is performed by hardware after reception of the slave address and direction bit. The slave address byte is the first byte received after the master generates a START condition. The slave address byte contains the 7-bit DS3231 address, which is 1101000, followed by the direction bit (R/W), which is 1 for a read. After receiving and decoding the slave address byte, the DS3231 outputs an acknowledge on SDA. The DS3231 then begins to transmit data starting with the register address pointed to by the register pointer. If the register pointer is not written to before the initiation of a read mode, the first address that is read is the last one stored in the register pointer. The DS3231 must receive a not acknowledge to end a read.

DS3231

Extremely Accurate I2C-Integrated RTC/TCXO/Crystal

Handling, PCB Layout, and Assembly

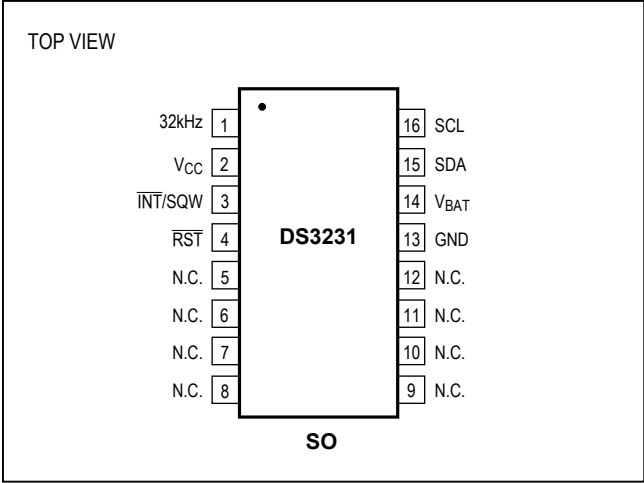
The DS3231 package contains a quartz tuning-fork crystal. Pick-and-place equipment can be used, but precautions should be taken to ensure that excessiveshocksareavoided.Ultrasoniccleaningshouldbe avoided to prevent damage to the crystal.

Avoid running signal traces under the package, unless a ground plane is placed between the package and the

signal line. All N.C. (no connect) pins must be connected to ground.

Moisture-sensitive packages are shipped from the factory dry packed. Handling instructions listed on the package label must be followed to prevent damage during reflow. Refer to the IPC/JEDEC J-STD-020 standard for moisture-sensitive device (MSD) classifications and reflow profiles. Exposure to reflow is limited to 2 times maximum.

Pin Configuration



Chip Information

SUBSTRATE CONNECTED TO GROUND

PROCESS: CMOS

Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
DS3231S#	0°C to +70°C	16 SO
DS3231SN#	-40°C to +85°C	16 SO

#Denotes an RoHS-compliant device that may include lead (Pb) that is exempt under RoHS requirements. The lead finish is JESD97 category e3, and is compatible with both lead-based and lead-free soldering processes. A “#” anywhere on the top mark denotes an RoHS-compliant device.

Package Information

For the latest package outline information and land patterns (footprints), go to www.maximintegrated.com/packages. Note that a “+”, “#”, or “-” in the package code indicates RoHS status only. Package drawings may show a different suffix character, but the drawing pertains to the package regardless of RoHS status.

PACKAGE TYPE	PACKAGE CODE	OUTLINE NO.	LAND PATTERN NO.
16 SO	W16#H2	21-0042	90-0107

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	1/05	Initial release.	—
1	2/05	Changed Digital Temp Sensor Output from $\pm 2^{\circ}\text{C}$ to $\pm 3^{\circ}\text{C}$.	1, 3
		Updated <i>Typical Operating Circuit</i> .	1
		Changed $T_A = -40^{\circ}\text{C}$ to $+85^{\circ}\text{C}$ to $T_A = T_{\text{MIN}}$ to T_{MAX} .	2, 3, 4
		Updated <i>Block Diagram</i> .	8
2	6/05	Added “UL Recognized” to Features; added lead-free packages and removed S from top mark info in <i>Ordering Information</i> table; added ground connections to the N.C. pin in the <i>Typical Operating Circuit</i> .	1
		Added “noncondensing” to operating temperature range; changed V_{PF} MIN from 2.35V to 2.45V.	2
		Added aging offset specification.	3
		Relabeled TOC4.	7
		Added arrow showing input on X1 in the <i>Block Diagram</i> .	8
		Updated pin descriptions for V_{CC} and V_{BAT} .	9
		Added the I ² C Interface section.	10
		Figure 1: Added sign bit to aging and temperature registers; added MSB and LSB.	11
		Corrected title for rate select bits frequency table.	13
		Added note that frequency stability over temperature spec is with aging offset register = 00h; changed bit 7 from Data to Sign (Crystal Aging Offset Register).	14
		Changed bit 7 from Data to Sign (Temperature Register); correct pin definitions in <i>I²C Serial Data Bus</i> section.	15
		Modified the <i>Handling</i> , <i>PC Board Layout</i> , and <i>Assembly</i> section to refer to J-STD-020 for reflow profiles for lead-free and leaded packages.	17
3	11/05	Changed lead-free packages to RoHS-compliant packages.	1
4	10/06	Changed $\overline{\text{RST}}$ and UL bullets in <i>Features</i> .	1
		Changed EC condition “ $V_{\text{CC}} > V_{\text{BAT}}$ ” to “ $V_{\text{CC}} = \text{Active Supply}$ (see Table 1).”	2, 3
		Modified Note 12 to correct t_{REC} operation.	6
		Added various conditions text to TOCs 1, 2, and 3.	7
		Added text to pin descriptions for 32kHz, V_{CC} , and $\overline{\text{RST}}$.	9
		Table 1: Changed column heading “Powered By” to “Active Supply”; changed “applied” to “exceeds V_{PF} ” in the <i>Power Control</i> section.	10
		Indicated BBSQW applies to both SQW and interrupts; simplified temp convert description (bit 5); added “output” to INT/SQW (bit 2).	13
		Changed the <i>Crystal Aging</i> section to the <i>Aging Offset</i> section; changed “this bit indicates” to “this bit controls” for the enable 32kHz output bit.	14
5	4/08	Added Warning note to EC table notes; updated Note 12.	6
		Updated the <i>Typical Operating Characteristics</i> graphs.	7
		In the <i>Power Control</i> section, added information about the POR state of the time and date registers; in the <i>Real-Time Clock</i> section, added to the description of the RST function.	10
		In Figure 1, corrected the months date range for 04h from 00–31 to 01–31.	11

Revision History (continued)

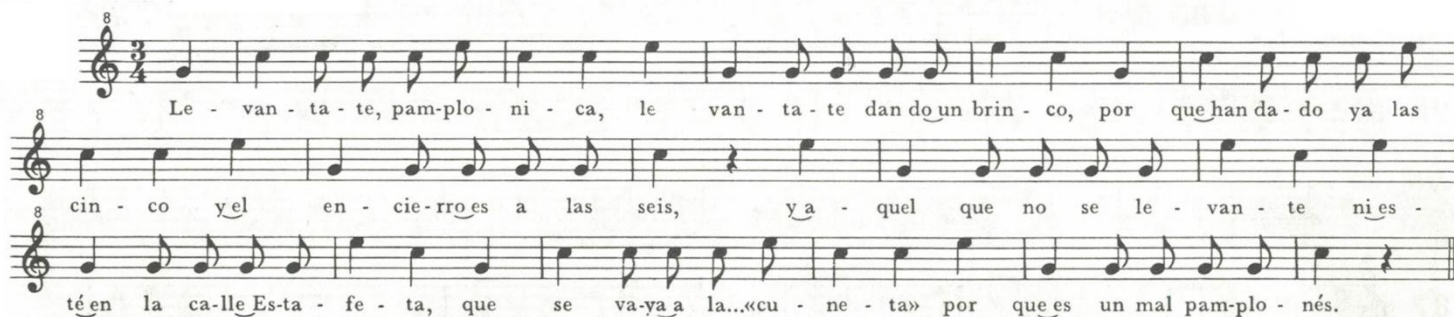
REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
6	10/08	Updated the <i>Typical Operating Circuit</i> .	1
		Removed the V_{PU} parameter from the <i>Recommended DC Operating Conditions</i> table and added verbiage about the pullup to the <i>Pin Description</i> table for INT/SQW, SDA, and SCL.	2, 9
		Added the Delta Time and Frequency vs. Temperature graph in the <i>Typical Operating Characteristics</i> section.	7
		Updated the <i>Block Diagram</i> .	8
		Added the V_{BAT} Operation section, improved some sections of text for the 32kHz TCXO and Pushbutton Reset Function sections.	10
		Added the register bit POR values to the register tables.	13, 14, 15
		Updated the <i>Aging Offset</i> and <i>Temperature Registers (11h–12h)</i> sections.	14, 15
		Updated the I ² C timing diagrams (Figures 3, 4, and 5).	16, 17
7	3/10	Removed the “S” from the top mark in the <i>Ordering Information</i> table and the <i>Pin Configuration</i> to match the packaging engineering marking specification.	1, 18
8	7/10	Updated the <i>Typical Operating Circuit</i> ; removed the “Top Mark” column from the <i>Ordering Information</i> ; in the <i>Absolute Maximum Ratings</i> section, added the theta-JA and theta-JC thermal resistances and Note 1, and changed the soldering temperature to +260°C (lead(Pb)-free) and +240°C (leaded); updated the functional description of the V_{BAT} pin in the <i>Pin Description</i> ; changed the timekeeping registers 02h, 09h, and 0Ch to “20 Hour” in Bit 5 of Figure 1; updated the BBSQW bit description in the <i>Control Register (0Eh)</i> section; added the land pattern no. to the <i>Package Information</i> table.	1, 2, 3, 4, 6, 9, 11, 12, 13, 18
9	1/13	Updated <i>Absolute Maximum Ratings</i> , and last paragraph in <i>Power Control</i> section	2, 10
10	3/15	Revised <i>Benefits and Features</i> section.	1

For pricing, delivery, and ordering information, please contact Maxim Direct at 1-888-629-4642, or visit Maxim Integrated's website at www.maximintegrated.com.

Maxim Integrated cannot assume responsibility for use of any circuitry other than circuitry entirely embodied in a Maxim Integrated product. No circuit patent licenses are implied. Maxim Integrated reserves the right to change the circuitry and specifications without notice at any time. The parametric values (min and max limits) shown in the *Electrical Characteristics* table are guaranteed. Other parametric values quoted in this data sheet are provided for guidance.

8.2 Anexo 2: Partituras de las canciones del Reloj Despertador

Canción 1: “Diana de San Fermín”



Fuente: Mönkemeyer, H. (1966). Método para tocar la flauta dulce soprano

Canción 2: “Las mañanitas”



Fuente: <http://www.tocapartituras.com/>

Canción 3: “Quinto levanta”



Fuente: <http://guitarrasediles.blogcindario.com>